PBS Professional® 11.1

User's Guide



PBS Works™

PBS Works is a division of Altair

PBS Professional User's Guide, Altair PBS Professional 11.1, Updated: 7/1/11. Edited by: Anne Urban

Copyright © 2003-2011 Altair Engineering, Inc. All rights reserved.

PBS™, PBS Works™, PBS GridWorks®, PBS Professional®, PBS Analytics™, PBS Catalyst™, e-Compute™, and e-Render™ are trademarks of Altair Engineering, Inc. and are protected under U.S. and international laws and treaties. All other marks are the property of their respective owners.

ALTAIR ENGINEERING INC. Proprietary and Confidential. Contains Trade Secret Information. Not for use or disclosure outside ALTAIR and its licensed clients. Information contained herein shall not be decompiled, disassembled, duplicated or disclosed in whole or in part for any purpose. Usage of the software is only as explicitly permitted in the end user software license agreement.

Copyright notice does not imply publication.

For more information, contact Altair at:
Web: www.pbsworks.com
Email: pbssales@altair.com

Technical Support

Location	Telephone	e-mail
North America	+1 248 614 2425	pbssupport@altair.com
China	+86 (0)21 6117 1666	es@altair.com.cn
France	+33 (0)1 4133 0992	francesupport@altair.com
Germany	+49 (0)7031 6208 22	hwsupport@altair.de
India	+91 80 66 29 4500	pbs-support@india.altair.com
Italy	+39 0832 315573 +39 800 905595	support@altairengineering.it
Japan	+81 3 5396 2881	pbs@altairjp.co.jp
Korea	+82 31 728 8600	support@altair.co.kr
Scandinavia	+46 (0)46 286 2050	support@altair.se
UK	+44 (0)1926 468 600	pbssupport@uk.altair.com

This document is proprietary information of Altair Engineering, Inc.

Table of Contents

Acknowledgements ix

About PBS Documentation xi

1 Concepts and Components 1

1.1 PBS Components 2

2 Getting Started With PBS 5

2.1 New Features in PBS Professional 11.1 6
2.2 New Features in PBS Professional 11.0 6
2.3 New Features in PBS Professional 10.4 7
2.4 New Features in PBS Professional 10.2 7
2.5 New Features in Version 10.1 8
2.6 New Features in Recent Releases 9
2.7 Deprecations 10
2.8 Backward Compatibility 10
2.9 Using PBS 10
2.10 PBS Interfaces 11
2.11 User's PBS Environment 13

2.12 Usernames Under PBS 14
2.13 Setting Up Your UNIX/Linux Environment 14
2.14 Setting Up Your Windows Environment 16
2.15 Environment Variables 20
2.16 Temporary Scratch Space: TMPDIR 22

3 Submitting a PBS Job 23

3.1 Vnodes: Virtual Nodes 23
3.2 PBS Resources 24
3.3 PBS Jobs 27
3.4 Submitting a PBS Job 33
3.5 Requesting Resources 36
3.6 Placing Jobs on Vnodes 50
3.7 Submitting Jobs Using Select & Place: Examples . . 58
3.8 Backward Compatibility 64
3.9 How PBS Parses a Job Script 67
3.10 A Sample PBS Jobs 68
3.11 Changing the Job's PBS Directive 70
3.12 Windows Jobs 71
3.13 Job Submission Options 74
3.14 Failed Jobs 92

4 Using the xpbs GUI 93

4.1 Using the `xpbs` command 93
4.2 Using xpbs: Definitions of Terms 95
4.3 Introducing the xpbs Main Display 96
4.4 Setting xpbs Preferences 104
4.5 Relationship Between PBS and xpbs 105
4.6 How to Submit a Job Using xpbs 106
4.7 Exiting xpbs 110
4.8 The xpbs Configuration File 110
4.9 xpbs Preferences 110

5 Working With PBS Jobs 115

5.1 Modifying Job Attributes 115
5.2 Holding and Releasing Jobs 119
5.3 Deleting Jobs 122
5.4 Sending Messages to Jobs 123
5.5 Sending Signals to Jobs 124
5.6 Changing Order of Jobs 125
5.7 Moving Jobs Between Queues 126
5.8 Converting a Job into a Reservation Job 128
5.9 Using Job History Information 129

6 Checking Job / System Status 133

6.1 The `qstat` Command 133
6.2 Viewing Job / System Status with xpbs 153
6.3 The qselect Command 154
6.4 Selecting Jobs Using xpbs 155
6.5 Using xpbs TrackJob Feature 156

7 Advanced PBS Features 159

7.1 UNIX Job Exit Status 159
7.2 Changing UNIX Job umask 160
7.3 Requesting qsub Wait for Job Completion 160
7.4 Specifying Job Dependencies 161
7.5 Delivery of Output Files 165
7.6 Input/Output File Staging 166
7.7 The pbsdsh Command 180
7.8 Advance and Standing Reservation of Resources . 181
7.9 Dedicated Time 199
7.10 Using Comprehensive System Accounting 200
7.11 Running PBS in a UNIX DCE Environment 201
7.12 Running PBS in a UNIX Kerberos Environment . . 202
7.13 Support for Large Page Mode on AIX 203
7.14 Checking License Availability 204

8 Job Arrays 205

8.1 Definitions 205
8.2 qsub: Submitting a Job Array 208
8.3 Job Array Attributes 209
8.4 Job Array States 210
8.5 PBS Environmental Variables 210
8.6 File Staging 210
8.7 PBS Commands 216
8.8 Other PBS Commands Supported for Job Arrays . 224
8.9 Job Arrays and xpbs 225
8.10 More on Job Arrays 225
8.11 Job Array Caveats 228

9 Multiprocessor Jobs 229

9.1 Job Placement 229
9.2 Submitting SMP Jobs 230
9.3 Submitting MPI Jobs 230
9.4 OpenMP Jobs with PBS 232
9.5 Hybrid MPI-OpenMP Jobs 232
9.6 MPI Jobs with PBS 235
9.7 MPI Jobs on the Altix 269
9.8 PVM Jobs with PBS 271
9.9 Checkpointing SGI MPI Jobs 272

10 HPC Basic Profile Jobs 273

10.1 Definitions 273
10.2 How HPC Basic Profile Jobs Work 274
10.3 Environmental Requirements for HPCBP 275
10.4 Submitting HPC Basic Profile Jobs 276
10.5 Managing HPCBP Jobs 282
10.6 Errors, Logging and Troubleshooting 283
10.7 Advice and Caveats 289
10.8 See Also 291

11 Submitting Cray Jobs 293

11.1 Introduction 293
11.2 PBS Jobs on the Cray 293
11.3 PBS Resources for the Cray 294
11.4 Rules for Submitting Jobs on the Cray 302
11.5 Techniques for Submitting Cray Jobs 305
11.6 Viewing Cray Job Information 311
11.7 Caveats and Advice 314
11.8 Errors and Logging 317

12 Using Provisioning 321
12.1 Definitions 321
12.2 How Provisioning Works 322
12.3 Requirements and Restrictions 325
12.4 Using Provisioning 326
12.5 Caveats and Errors 328

Appendix A: Converting NQS to PBS 331
13.1 Converting Date Specifications 332

Appendix B: License Agreement 333

Index 347

Table of Contents

Acknowledgements

PBS Professional is the enhanced commercial version of the PBS software originally developed for NASA. The NASA version had a number of corporate and individual contributors over the years, for which the PBS developers and PBS community is most grateful. Below we provide formal legal acknowledgements to corporate and government entities, then special thanks to individuals.

The NASA version of PBS contained software developed by NASA Ames Research Center, Lawrence Livermore National Laboratory, and MRJ Technology Solutions. In addition, it included software developed by the NetBSD Foundation, Inc., and its contributors as well as software developed by the University of California, Berkeley and its contributors.

Other contributors to the NASA version of PBS include Bruce Kelly and Clark Streeter of NERSC; Kent Crispin and Terry Heidelberg of LLNL; John Kochmar and Rob Pennington of *Pittsburgh Supercomputing Center*; and Dirk Grunwald of *University of Colorado, Boulder*. The ports of PBS to the Cray T3e and the IBM SP SMP were funded by *DoD USAERDC*; the port of PBS to the Cray SV1 was funded by *DoD MSIC*.

No list of acknowledgements for PBS would possibly be complete without special recognition of the first two beta test sites. Thomas Milliman of the *Space Sciences Center* of the *University of New Hampshire* was the first beta tester. Wendy Lin of *Purdue University* was the second beta tester and holds the honor of submitting more problem reports than anyone else outside of NASA.

About PBS Documentation

Where to Keep the Documentation

To make cross-references work, put all of the PBS guides in the same directory.

What is PBS Professional?

PBS is a workload management system that provides a unified batch queuing and job management interface to a set of computing resources.

The PBS Professional Documentation

The documentation for PBS Professional includes the following:

PBS Professional Administrator's Guide

Provides the PBS administrator with the information required to configure and manage PBS Professional (PBS).

PBS Professional Quick Start Guide:

Provides a quick overview of PBS Professional installation and license file generation.

PBS Professional Installation & Upgrade Guide:

Contains information on installing and upgrading PBS Professional.

PBS Professional User's Guide:

Covers user commands and how to submit, monitor, track, delete, and manipulate jobs.

PBS Professional Programmer's Guide:

Discusses the PBS application programming interface (API).

PBS Professional Reference Guide:

Contains PBS reference material.

PBS Manual Pages:

Describe PBS commands, resources, attributes, APIs

Ordering Software and Publications

To order additional copies of this manual and other PBS publications, or to purchase additional software licenses, contact your Altair sales representative. Contact information is included on the copyright page of this book.

Document Conventions

PBS documentation uses the following typographic conventions:

`abbr`eviation

The shortest acceptable abbreviation of a command or subcommand is underlined.

`command`

Commands such as `qmgr` and `scp`

`input`

Command-line instructions

`manpage(x)`

File and path names. Manual page references include the section number in parentheses appended to the manual page name.

formats

Formats

Attributes

Attributes, parameters, objects, variable names, resources, types

Values

Keywords, instances, states, values, labels

Definitions

Terms being defined

`Output`

Output or example code

File contents

Chapter 1

Concepts and Components

PBS is a distributed workload management system. As such, PBS handles the management and monitoring of the computational workload on a set of one or more computers. Modern workload management solutions like PBS Professional include the features of traditional batch queueing but offer greater flexibility and control than first generation batch systems (such as NQS).

Workload management systems have three primary roles:

Queuing

The collecting together of work or tasks to be run on a computer. Users submit tasks or "jobs" to the resource management system where they are queued up until the system is ready to run them.

Scheduling

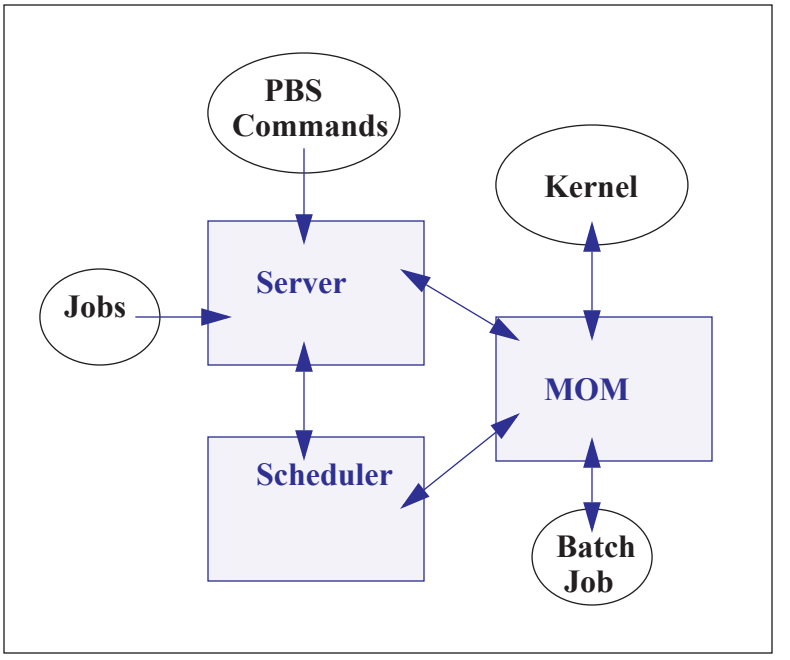
The process of selecting which jobs to run, when, and where, according to a predetermined policy. Sites balance competing needs and goals on the system(s) to maximize efficient use of resources (both computer time and people time).

Monitoring

The act of tracking and reserving system resources and enforcing usage policy. This includes both software enforcement of usage limits and user or administrator monitoring of scheduling policies to see how well they are meeting stated goals.

1.1 PBS Components

PBS consist of two major component types: user-level commands and system daemons/services. A brief description of each is given here to help you understand how the pieces fit together, and how they affect you.



Commands

PBS supplies both command line programs that are POSIX 1003.2d conforming and a graphical interface. These are used to submit, monitor, modify, and delete jobs. These *client commands* can be installed on any system type supported by PBS and do not require the local presence of any of the other components of PBS.

There are three command classifications: user commands, which any authorized user can use, operator commands, and manager (or administrator) commands. Operator and manager commands which require specific access privileges are discussed in the **PBS Professional Administrator's Guide**.

Server

The *Job Server* daemon/service is the central focus for PBS. Within this document, it is generally referred to as *the Server* or by the execution name *pbs_server*. All commands and the other daemons/services communicate with the Server via an *Internet Protocol* (IP) network. The Server's main function is to provide the basic batch services such as receiving/creating a batch job, modifying the job, and running the job. Normally, there is one Server managing a given set of resources. However if the Server Failover feature is enabled, there will be two Servers.

Job Executor (MOM)

The *Job Executor* or *MOM* is the daemon/service which actually places the job into execution. This process, *pbs_mom,* is informally called *MOM* as it is the mother of all executing jobs. (MOM is a reverse-engineered acronym that stands for Machine Oriented Mini-server.) MOM places a job into execution when it receives a copy of the job from a Server. MOM creates a new session that is as identical to a user login session as is possible. (For example under UNIX, if the user's login shell is `csh`, then MOM creates a session in which `.login` is run as well as `.cshrc`.) MOM also has the responsibility for returning the job's output to the user when directed to do so by the Server. One MOM runs on each computer which will execute PBS jobs.

Scheduler

The *Job Scheduler* daemon/service, *pbs_sched*, implements the site's policy controlling when each job is run and on which resources. The Scheduler communicates with the various MOMs to query the state of system resources and with the Server for availability of jobs to execute. The interface to the Server is through the same API as used by the client commands. Note that the Scheduler interfaces with the Server with the same privilege as the PBS manager.

Chapter 2

Getting Started With PBS

This chapter introduces the user to PBS Professional. It describes new user-level features in this release, explains the different user interfaces, introduces the concept of a PBS “job”, and shows how to set up your environment for running batch jobs with PBS.

2.1 New Features in PBS Professional 11.1

2.1.1 Support for Interlagos Hardware

You can request Interlagos hardware for your jobs. See section 11.5.10 “Requesting Interlagos Hardware” on page 311.

2.2 New Features in PBS Professional 11.0

2.2.1 Improved Cray Integration

PBS is more tightly integrated with Cray systems. You can use the PBS select and place language when submitting Cray jobs. See section “Submitting Cray Jobs” on page 293.

2.2.2 Enhanced Job Placement

PBS allows job submitters to scatter chunks by vnode in addition to scattering by host. PBS also allows job submitters to reserve entire hosts via a job’s placement request. See section 3.6 “Placing Jobs on Vnodes” on page 50.

2.3 New Features in PBS Professional 10.4

2.3.1 Estimated Job Start Times

PBS can estimate the start time and vnodes for jobs. See section 6.1.21 “Viewing Estimated Start Times For Jobs” on page 152.

2.3.2 Unified Job Submission

PBS allows users to submit jobs using the same scripts, whether the job is submitted on a Windows or UNIX/Linux system. See section 3.3.3.1 “Python Job Scripts” on page 29.

2.4 New Features in PBS Professional 10.2

2.4.1 Provisioning

PBS provides automatic provisioning of an OS or application on vnodes that are configured to be provisioned. When a job requires an OS that is available but not running, or an application that is not installed, PBS provisions the vnode with that OS or application. See Chapter 12, "Using Provisioning", on page 321Chapter 12, "Using Provisioning", on page 321.

2.4.2 Walltime as Checkpoint Interval Measure

PBS allows a job to be checkpointed according to its walltime usage. See the `pbs_job_attributes(7B)` manual page.

2.4.3 Employing User Space Mode on IBM InfiniBand Switches

PBS allows users submitting POE jobs to use InfiniBand switches in User Space mode. See section 9.6.3 “MPI Jobs Using AIX, POE” on page 236.

2.5 New Features in Version 10.1

2.5.1 Submitting HPCBP Jobs

PBS Professional can schedule and manage jobs on one or more HPC Basic Profile compliant servers using the Grid Forum OGSA HPC Basic Profile web services standard. You can submit a generic job to PBS, so that PBS can run it on an HPC Basic Profile Server. This chapter describes how to use PBS for HPC Basic Profile jobs. See Chapter 10, "HPC Basic Profile Jobs", on page 273.

2.5.2 Using Job History Information

PBS Professional can provide job history information, including what the submission parameters were, whether the job started execution, whether execution succeeded, whether staging out of results succeeded, and which resources were used. PBS can keep job history for jobs which have finished execution, were deleted, or were moved to another server. See section 5.9 “Using Job History Information” on page 129.

2.5.3 Reservation Fault Tolerance

PBS attempts to reconfirm reservations for which associated vnodes have become unavailable. See section 7.8.8.1.1 “Reservation Fault Tolerance” on page 195.

2.6 New Features in Recent Releases

2.6.1 Path to Binaries (10.0)

The path to the PBS binaries may have changed for your system. If the old path was not one of `/opt/pbs`, `/usr/pbs`, or `/usr/local/pbs`, you may need to add `/opt/pbs/default` to your PATH environment variable.

2.6.2 Using job_sort_key (10.0)

The sort_priority option to job_sort_key is replaced with the job_priority option.

2.6.3 Job-Specific Staging and Execution Directories (9.2)

PBS can now provide a staging and execution directory for each job. Jobs have new attributes `sandbox` and `jobdir`, the MOM has a new option `$jobdir_root`, and there is a new environment variable called `PBS_JOBDIR`. If the job's `sandbox` attribute is set to `PRIVATE`, PBS creates a job-specific staging and execution directory. If the job's `sandbox` attribute is unset or is set to `HOME`, PBS uses the user's home directory for staging and execution, which is how previous versions of PBS behaved. See section 7.6 "Input/Output File Staging" on page 166.

2.6.4 Standing Reservations (9.2)

PBS now provides a facility for making standing reservations. A standing reservation is a series of advance reservations. The pbs_rsub command is used to create both advance and standing reservations. See section 7.8 "Advance and Standing Reservation of Resources" on page 181.

2.7 Deprecations

For a list of deprecations, see section 1.3, "Deprecations" on page 10 in the PBS Professional Administrator's Guide.

2.8 Backward Compatibility

2.8.1 Job Dependencies Affected By Job History

Enabling job history changes the behavior of dependent jobs. If a job j1 depends on a finished job j2 for which PBS is maintaining history than j1 will go into the held state. If job j1 depends on a finished job j3 that has been purged from the historical records than j1 will be rejected just as in previous versions of PBS where the job was no longer in the system.

2.8.2 PBS path information no longer saved in AUTOEXEC.BAT

Any value for PATH saved in AUTOEXEC.BAT may be lost after installation of PBS. If there is any path information that needs to be saved, AUTOEXEC.BAT must be edited by hand after the installation of PBS. PBS path information is no longer saved in AUTOEXEC.BAT.

2.9 Using PBS

From the user's perspective, a workload management system allows you to make more efficient use of your time. You specify the tasks you need executed. The system takes care of running these tasks and returning the

results to you. If the available computers are full, then the workload management system holds your work and runs it when the resources are available.

With PBS you create a *batch job* which you then submit to PBS. A batch job is a file (a shell script under UNIX or a `cmd` batch file under Windows) containing the set of commands you want to run on some set of execution machines. It also contains directives which specify the characteristics (attributes) of the job, and resource requirements (e.g. memory or CPU time) that your job needs. Once you create your PBS job, you can reuse it if you wish. Or, you can modify it for subsequent runs. For example, here is a simple PBS batch job:

UNIX:

```
#!/bin/sh
#PBS -l walltime=1:00:00
#PBS -l mem=400mb,ncpus=4
./my_application
```

Windows:

```
#PBS -l walltime=1:00:00
#PBS -l mem=400mb,ncpus=4
my_application
```

Don't worry about the details just yet; the next chapter will explain how to create a batch job of your own.

2.10 PBS Interfaces

PBS provides two user interfaces: a command line interface (CLI) and a graphical user interface (GUI). The CLI lets you type commands at the system prompt. The GUI is a graphical point-and-click interface. The "user commands" are discussed in this book; the "administrator commands" are discussed in the **PBS Professional Administrator's Guide**. The subsequent chapters of this book will explain how to use both the CLI and GUI versions of the user commands to create, submit, and manipulate PBS jobs.

.

Table 2-1: PBS Professional User and Manager Commands

User Commands		Administrator Commands	
Command	Purpose	Command	Purpose
nqs2pbs	Convert from NQS	pbs-report	Report job statistics
pbs_rdel	Delete a Reservation		
pbs_rstat	Status a Reservation	pbs_hostn	Report host name(s)
pbs_ password	Update per user / per server password[1]	pbs_migrate _users	Migrate per user / per server passwords [1]
pbs_rsub	Submit a Reservation	pbs_probe	PBS diagnostic tool
pbsdsh	PBS distributed shell		
qalter	Alter job	pbs_tclsh	TCL with PBS API
qdel	Delete job	pbsfs	Show fairshare usage
qhold	Hold a job	pbsnodes	Vnode manipulation
qmove	Move job	printjob	Report job details
qmsg	Send message to job	qdisable	Disable a queue
qorder	Reorder jobs	qenable	Enable a queue
qrls	Release hold on job	qmgr	Manager interface
qselect	Select jobs by criteria	qrerun	Requeue running job
qsig	Send signal to job	qrun	Manually start a job

Table 2-1: PBS Professional User and Manager Commands

User Commands		Administrator Commands	
qstat	Status job, queue, Server	qstart	Start a queue
qsub	Submit a job	qstop	Stop a queue
tracejob	Report job history	qterm	Shutdown PBS
xpbs	Graphical User Interface	xpbsmon	GUI monitoring tool

Notes:

1 Available on Windows only.

2.11 User's PBS Environment

In order to have your system environment interact seamlessly with PBS, there are several items that need to be checked. In many cases, your system administrator will have already set up your environment to work with PBS.

In order to use PBS to run your work, the following are needed:

- User must have access to the resources/hosts that the site has configured for PBS
- User must have a valid account (username and group) on the execution hosts
- User must be able to transfer files between hosts (e.g. via `rcp` or `scp`)
- User's time zone environment variable must be set correctly in order to use advance and standing reservations. See section 7.8.9.1 "Setting the Submission Host's Time Zone" on page 196.

The subsequent sections of this chapter discuss these requirements in detail, and provide various setup procedures.

2.12 Usernames Under PBS

By default PBS will use your login identifier as the username under which to run your job. This can be changed via the "`-u`" option to `qsub`. See section 3.13.14 "Specifying Job User ID" on page 87. The user submitting the job must be authorized to run the job under the execution user name (whether explicitly specified or not).

IMPORTANT:

PBS enforces a maximum username length of 15 characters. If a job is submitted to run under a username longer than this limit, the job will be rejected.

2.13 Setting Up Your UNIX/Linux Environment

2.13.1 Setting PBS_EXEC on UNIX/Linux

In order to make it easier to submit a job script, you can set up your environment so that the correct value for `PBS_EXEC` is used automatically.

Under `sh` or `bash`, do the following:

```
% . /etc/pbs.conf
```

2.13.2 Preventing Problems

A user's job may not run if the user's start-up files (i.e `.cshrc`, `.login`, or `.profile`) contain commands which attempt to set terminal characteristics. Any such command sequence within these files should be skipped by testing for the environment variable **`PBS_ENVIRONMENT`**. This can be done as shown in the following sample `.login`:

```
setenv MANPATH /usr/man:/usr/local/man:$MANPATH
if ( ! $?PBS_ENVIRONMENT ) then
      do terminal settings here
endif
```

You should also be aware that commands in your startup files should not generate output when run under PBS. As in the previous example, commands that write to stdout should not be run for a PBS job. This can be done as shown in the following sample `.login`:

```
setenv MANPATH /usr/man:/usr/local/man:$MANPATH
if ( ! $?PBS_ENVIRONMENT ) then
      do terminal settings here
      run command with output here
endif
```

When a PBS job runs, the "exit status" of the last command executed in the job is reported by the job's shell to PBS as the "exit status" of the job. (We will see later that this is important for job dependencies and job chaining.) However, the last command executed might not be the last command in your job. This can happen if your job's shell is `csh` on the execution host and you have a `.logout` there. In that case, the last command executed is from the `.logout` and not your job. To prevent this, you need to preserve the job's exit status in your `.logout` file, by saving it at the top, then doing an explicit `exit` at the end, as shown below:

```
set EXITVAL = $status
previous contents of .logout here
exit $EXITVAL
```

Likewise, if the user's login shell is `csh` the following message may appear in the standard output of a job:

Warning: no access to tty, thus no job control in this shell

This message is produced by many `csh` versions when the shell determines that its input is not a terminal. Short of modifying `csh`, there is no way to eliminate the message. Fortunately, it is just an informative message and has no effect on the job.

2.13.2.1 Interactive Jobs

An interactive job comes complete with a pseudotty suitable for running those commands that set terminal characteristics. But more importantly, it does not caution the user that starting something in the background that would persist after the user has exited from the interactive environment might cause trouble for some moms. They could believe that once the interactive session terminates, all the user's processes are gone with it. For example, applications like ssh-agent background themselves into a new session and would prevent a CPU set-enabled mom from deleting the CPU set for the job. This in turn might cause subsequent failed attempts to run new jobs, resulting in them being placed in a held state.

2.13.3 Setting MANPATH on SGI Systems

The PBS "man pages" (UNIX manual entries) are installed on SGI systems under `/usr/bsd`, or for the Altix, in `/usr/pbs/man`. In order to find the PBS man pages, users will need to ensure that `/usr/bsd` is set within their `MANPATH`. The following example illustrates this for the C shell:

```
setenv MANPATH /usr/man:/usr/local/man:/usr/ bsd:$MAN-
   PATH
```

2.14 Setting Up Your Windows Environment

This section discusses the setup steps needed for running PBS Professional in a Microsoft Windows environment, including host and file access, passwords, and restrictions on home directories.

2.14.1 Setting PBS_EXEC on Windows

In order to make it easier to submit a job script, you can set up your environment so that the correct value for PBS_EXEC is used automatically. Under Windows, do the following:

1. Look into `"C:\Program Files\PBS Pro\pbs.conf"`, and get the value of `PBS_EXEC`. It will be something like "`C:\Program Files\PBS Pro\exec`".
2. Set your environment accordingly:

   `cmd> set PBS_EXEC="<path>"`

   For example,

   `cmd> set PBS_EXEC="C:\Program Files\PBS Pro\exec"`

2.14.2 Windows User's HOMEDIR

Each Windows user is assumed to have a home directory (`HOMEDIR`) where his/her PBS jobs are initially started.

If a user has not been explicitly assigned a home directory, then PBS will use this Windows-assigned default as the base location for the user's default home directory. More specifically, the actual home path will be:

```
[PROFILE_PATH]\My Documents\PBS Pro
```

For instance, if a *userA* has not been assigned a home directory, it will default to a local home directory of:

```
\Documents and Settings\userA\My
 Documents\PBS Pro
```

UserA's job will use the above path as its working directory.

Note that Windows can return as PROFILE_PATH one of the following forms:

\Documents and Settings*username*

\Documents and Settings*username*.*local-host name*

\Documents and Settings*username*.*local-host name*.00*N*

where N is a number

\Documents and Settings*username*.*domain-name*

2.14.3 Windows Usernames and Job Submission

A PBS job is run from a user account and the associated username string must conform to the POSIX-1 standard for portability. That is, the username must contain only alphanumeric characters, dot (.), underscore (_), and/or hyphen "-". The hyphen must not be the first letter of the username. If "@" appears in the username, then it will assumed to be in the context of a Windows domain account: username@domainname. An exception to the above rule is the space character, which is allowed. If a space character appears in a username string, then it will be displayed quoted and must be specified in a quoted manner. The following example requests the job to run under account "Bob Jones".

```
qsub -u "Bob Jones" my_job
```

2.14.4 Windows rhosts File

The Windows rhosts file is located in the user's [PROFILE_PATH], for example: \Documents and Settings*username*\.rhosts, with the format:

hostname username

IMPORTANT:

Be sure the `.rhosts` file is owned by user or an administrator-type group, and has write access granted only to the owning user or an administrator or group.

This file can also determine if a remote user is allowed to submit jobs to the local PBS Server, if the mapped user is an Administrator account. For example, the following entry in user `susan`'s `.rhosts` file on the server would permit user `susan` to run jobs submitted from her workstation wks031:

```
wks031 susan
```

Furthermore, in order for Susan's output files from her job to be returned to her automatically by PBS, she would need to add an entry to her `.rhosts` file on her workstation naming the execution host `Host1`.

```
Host1 susan
```

If instead, Susan has access to several execution hosts, she would need to add all of them to her `.rhosts` file:

```
Host1 susan
Host2 susan
Host3 susan
```

Note that Domain Name Service (DNS) on Windows may return different permutations for a full hostname, thus it is important to list all the names that a host may be known. For instance, if Host4 is known as "Host4", "Host4.<subdomain>", or "Host4.<subdomain>.<domain>" you should list all three in the `.rhosts` file.

```
Host4 susan
Host4.subdomain susan
Host4.subdomain.domain susan
```

As discussed in the previous section, usernames with embedded white space must also be quoted if specified in any `hosts.equiv` or `.rhosts` files, as shown below.

```
Host5.subdomain.domain "Bob Jones"
```

2.14.5 Windows Mapped Drives and PBS

In Windows XP, when you map a drive, it is mapped "locally" to your session. The mapped drive cannot be seen by other processes outside of your session. A drive mapped on one session cannot be un-mapped in another session even if it's the same user. This has implications for running jobs under PBS. Specifically if you map a drive, `chdir` to it, and submit a job from that location, the vnode that executes the job may not be able to deliver the files back to the same location from which you issued `qsub`. The workaround is to use the "`-o`" or "`-e`" options to `qsub` and specify a local (non-mapped) directory location for the job output and error files. For details see section 3.13.2.2 "Specifying Path for Output and Error Files" on page 77.

2.15 Environment Variables

There are a number of environment variables provided to the PBS job. Some are taken from the user's environment and carried with the job. Others are created by PBS. Still others can be explicitly created by the user for exclusive use by PBS jobs. All PBS-provided environment variable names start with the characters "`PBS_`". Some are then followed by a capital O ("`PBS_O_`") indicating that the variable is from the job's originating environment (i.e. the user's). See "PBS Environment Variables" on page 517

of the PBS Professional Reference Guide for a full listing of all environment variables provided to PBS jobs and their meaning. The following short example lists some of the more useful variables, and typical values.

```
PBS_O_HOME=/u/user1
PBS_O_LOGNAME=user1
PBS_O_PATH=/usr/new/bin:/usr/local/bin:/bin
PBS_O_SHELL=/sbin/csh
PBS_O_HOST=cray1
PBS_O_WORKDIR=/u/user1
PBS_O_QUEUE=submit
PBS_JOBID=16386.cray1
PBS_QUEUE=crayq
PBS_ENVIRONMENT=PBS_INTERACTIVE
```

There are a number of ways that you can use these environment variables to make more efficient use of PBS. In the example above we see **PBS_ENVIRONMENT**, which we used earlier in this chapter to test if we were running under PBS. Another commonly used variable is **PBS_O_WORKDIR** which contains the name of the directory from which the user submitted the PBS job.

There are also two environment variables that you can set to affect the behavior of PBS. The environment variable **PBS_DEFAULT** defines the name of the default PBS Server. Typically, it corresponds to the system name of the host on which the Server is running. If **PBS_DEFAULT** is not set, the default is defined by an administrator established file (usually `/etc/pbs.conf` on UNIX, and `[PBS Destination Folder]\pbs.conf` on Windows).

The environment variable **PBS_DPREFIX** determines the prefix string which identifies directives in the job script. The default prefix string is "#PBS"; however the Windows user may wish to change this as discussed in section 3.11 "Changing the Job's PBS Directive" on page 70.

2.16 Temporary Scratch Space: TMPDIR

PBS creates an environment variable, TMPDIR, which contains the full path name to a temporary "scratch" directory created for each PBS job. The directory will be removed when the job terminates.

Under Windows, TMP will also be set to the value of %TMPDIR%. The temporary directory will be created under either \winnt\temp or \windows\temp, unless an alternative directory was specified by the administrator in the MOM configuration file.

Users can access the job-specific temporary space, by changing directory to it inside their job script. For example:

UNIX:

```
cd $TMPDIR
```

Windows:

```
cd %TMPDIR%
```

Chapter 3

Submitting a PBS Job

This chapter describes virtual nodes, how to submit a PBS job, how to use resources for jobs, how to place your job on vnodes, job attributes, and several related topics.

3.1 Vnodes: Virtual Nodes

A virtual node, or vnode, is an abstract object representing a set of resources which form a usable part of a machine. This could be an entire host, or a nodeboard or a blade. A single host can be made up of multiple vnodes. Each vnode can be managed and scheduled independently. PBS views hosts as being composed of one or more vnodes. Jobs run on one or more vnodes. See the `pbs_node_attributes(7B)` man page.

3.1.1 Relationship Between Hosts, Nodes, and Vnodes

A host is any computer. Execution hosts used to be called nodes. However, some machines such as the Altix can be treated as if they are made up of separate pieces containing CPUs, memory, or both. Each piece is called a vnode. Some hosts have a single vnode and some have multiple vnodes. PBS treats all vnodes alike in most respects. Chunks cannot be split across hosts, but they can be split across vnodes on the same host.

Resources that are defined at the host level are applied to vnodes. A host-level resource is shared among the vnodes on that host. This sharing is managed by the MOM.

3.1.2 Vnode Types

What were called nodes are now called vnodes. All vnodes are treated alike, and are treated the same as what were called “time-shared nodes”. The types “time-shared” and “cluster” are deprecated. The `:ts` suffix is deprecated. It is silently ignored, and not preserved during rewrite. The vnode attribute `ntype` is only used to distinguish between PBS and Globus vnodes. It is read-only.

3.2 PBS Resources

Resources can be available on the server and queues, and on vnodes. Jobs can request resources. Resources are allocated to jobs, and some resources such as memory are consumed by jobs. The scheduler matches requested resources with available resources, according to rules defined by the administrator. PBS can enforce limits on resource usage by jobs.

PBS provides built-in resources, and in addition, allows the administrator to define custom resources. The administrator can specify which resources are available on a given vnode, as well as at the server or queue level (e.g.

floating licenses.) Vnodes can share resources. The administrator can also specify default arguments for qsub. These arguments can include resources. See the `qsub(1B)` man page.

Resources made available by defining them via resources_available at the server level are only used as job-wide resources. These resources (e.g. walltime, server_dyn_res) are requested using -l RESOURCE=VALUE. Resources made available at the host (vnode) level are only used as chunk resources, and can only be requested within chunks using -l select=RESOURCE=VALUE. Resources such as mem and ncpus can only be used at the vnode level.

Resources are allocated to jobs both by explicitly requesting them and by applying specified defaults. Jobs explicitly request resources either at the vnode level in chunks defined in a selection statement, or in job-wide resource requests. See the `pbs_resources(7B)` manual page.

Jobs are assigned limits on the amount of resources they can use. These limits apply to how much the job can use on each vnode (per-chunk limit) and to how much the whole job can use (job-wide limit). Limits are derived from both requested resources and applied default resources.

Each chunk's per-chunk limits determine how much of any resource can be used in that chunk. Per-chunk resource usage limits are the amount of per-chunk resources requested, both from explicit requests and from defaults.

Job resource limits set a limit for per-job resource usage. Job resource limits are derived in this order from:

1. explicitly requested job-wide resources (e.g. -l resource=value)
2. the select specification (e.g. -l select =...)
3. the queue's resources_default.RES
4. the server's resources_default.RES
5. the queue's resources_max.RES
6. the server's resources_max.RES

The server's default_chunk.RES does **not** affect job-wide limits.

The resources requested for chunks in the select specification are summed, and this sum is used for a job-wide limit. Job resource limits from sums of all chunks override those from job-wide defaults and resource requests.

Various limit checks are applied to jobs. If a job's job resource limit exceeds queue or server restrictions, it will not be put in the queue or accepted by the server. If, while running, a job exceeds its limit for a consumable or time-based resource, it will be terminated.

A “consumable” resource is one that is reduced by being used, for example, ncpus, licenses, or mem. A “non-consumable” resource is not reduced through use, for example, walltime or a boolean resource.

Resources are tracked in server, queue, vnode and job attributes. Servers, queues and vnodes have two attributes, resources_available.RESOURCE and resources_assigned.RESOURCE. The resources_available.RESOURCE attribute tracks the total amount of the resource available at that server, queue or vnode, without regard to how much is in use. The resources_assigned.RESOURCE attribute tracks how much of that resource has been assigned to jobs at that server, queue or vnode. Jobs have an attribute called resources_used.RESOURCE which tracks the amount of that resource used by that job.

The administrator can set server and queue defaults for resources used in chunks. See the PBS Professional Administrator’s Guide and the pbs_server_attributes(7B) and pbs_queue_attributes(7B) manual pages.

For a thorough discussion of PBS resources, see section , "PBS Resources" on page 317 in the PBS Professional Administrator’s Guide.

3.2.1 Unset Resources

When job resource requests are being matched with available resources, a numerical resource that is unset on a host is treated as if it were zero, and an unset string cannot be matched. An unset Boolean resource is treated as if it is set to “False”. An unset resource at the server or queue is treated as if it were infinite.

3.2.2 Resource Names

The resource name is any string made up of alphanumeric characters, where the first character is alphabetic. Resource names must start with an alphabetic character and can contain alphanumeric, underscore (“_”), and dash (“-”) characters.

3.2.3 Specifying Resource Values

- Resource values which contain commas, quotes, plus signs, equal signs, colons, or parentheses must be quoted to PBS. The string must be enclosed in quotes so that the command (e.g. qsub, qalter) will parse it correctly.
- When specifying resources via the command line, any quoted strings must be escaped or enclosed in another set of quotes. This second set of quotes must be different from the first set, meaning that double quotes must be enclosed in single quotes, and vice versa.
- If a string resource value contains spaces or shell metacharacters, enclose the string in quotes, or otherwise escape the space and metacharacters. Be sure to use the correct quotes for your shell and the behavior you want.

3.2.4 Resource Types

See “Resource Data Types” on page 347 of the PBS Professional Reference Guide for a description of resource types.

3.2.5 Built-in Resources

See “Built-in Resources” on page 350 of the PBS Professional Reference Guide for a list of built-in resources.

3.3 PBS Jobs

3.3.1 Rules for Submitting Jobs

- The "place" specification cannot be used without the "select" specifica-

tion. See section 3.6 “Placing Jobs on Vnodes” on page 50.

- A "select" specification cannot be used with a "nodes" specification.
- A "select" specification cannot be used with old-style resource requests such as -lncpus, -lmem, -lvmem, -larch, -lhost.
- The built-in resource "software" is not a vnode-level resource. See “Built-in Resources” on page 350 of the PBS Professional Reference Guide.
- A PBS job can be submitted at the command line or via `xpbs`.
- At the command line, the user can create a job script, and submit it. During submission it is possible to override elements in the job script. Alternatively, PBS will read from input typed at the command line.

3.3.2 Introduction to the PBS Job Script

3.3.2.1 Contents of a Job Script

A PBS job script consists of:

- An optional shell specification
- PBS directives
- Tasks (programs or commands)

3.3.2.2 Types of Job Scripts

PBS allows you to use various kinds of job scripts. You can use any of the following:

- A Python script that can run under Windows or UNIX/Linux
- A UNIX shell script that runs under UNIX/Linux
- Windows command batch script under Windows

3.3.2.3 Submitting a Job Script

Before submitting a job script using these instructions, be sure to set your environment appropriately If you want the correct value for `PBS_EXEC` to be used automatically, see section 2.13.1 "Setting PBS_EXEC on UNIX/Linux" on page 14 and section 2.14.1 "Setting PBS_EXEC on Windows" on page 17.

To submit a PBS job, type the following:

UNIX/Linux shell script:

```
qsub <name of shell script>
```

UNIX/Linux Python script:

```
qsub -S $PBS_EXEC/bin/pbs_python <python job script>
```

Windows command script:

```
qsub <name of job script>
```

Windows Python script:

```
qsub -S %PBS_EXEC%\bin\pbs_python.exe <python job
   script>
```

If the path contains any spaces, it must be quoted, for example:

```
qsub -S "%PBS_EXEC%\bin\pbs_python.exe" <python job
   script>
```

3.3.3 The Job Script

3.3.3.1 Python Job Scripts

PBS allows you to submit jobs using a Python script. You can use the same Python script under Windows or UNIX/Linux. PBS includes a Python package, allowing Python job scripts to run; you do not need to install Python. To run a Python job script:

UNIX/Linux:

```
qsub -S $PBS_EXEC/bin/pbs_python <script name>
```

Windows:

```
qsub -S %PBS_EXEC%\bin\pbs_python.exe <script name>
```

If the path contains any spaces, it must be quoted, for example:

```
qsub -S "%PBS_EXEC%\bin\pbs_python.exe" <python job
   script>
```

You can include PBS directives in a Python job script as you would in a UNIX shell script. For example:

```
% cat myjob.py
#PBS -l select=1:ncpus=3:mem=1gb
#PBS -N HelloJob
print "Hello"
```

Python job scripts can access Win32 APIs, including the following modules:

- Win32api
- Win32con
- Pywintypes

3.3.3.1.1 Windows Python Caveat

If you have Python natively installed, and you need to use the `win32api`, make sure that you import `pywintypes` before `win32api`, otherwise you will get an error. Do the following:

```
cmd> pbs_python
>> import pywintypes
>> import win32api
```

3.3.3.2 UNIX Shell Scripts

Since the job file can be a shell script, the first line of a shell script job file specifies which shell to use to execute the script. The user's login shell is the default, but you can change this. This first line can be omitted if it is acceptable for the job file to be interpreted using the login shell.

3.3.3.3 Windows Command Scripts

If the job file is a shell script, specify the shell in the first line of the job file.

3.3.3.3.1 Windows Caveats

In Windows, if you use `notepad` to create a job script, the last line does not automatically get newline-terminated. Be sure to put one explicitly, otherwise, PBS job will get the following error message:

```
More?
```

when the Windows command interpreter tries to execute that last line.

3.3.3.4 Perl Scripts Under Windows

PBS jobs execute under the native cmd environment, and not under cygwin. To run a Perl script under Windows, you must specify the path to Perl in your job script.

Format:

<path to Perl> my_job.pl [arguments to script]

3.3.3.5 PBS Directives

PBS directives are at the top of the script file. They are used to request resources or set attributes. A directive begins with the default string "#PBS". Attributes can also be set using options to the `qsub` command, which will override directives. The limit for a PBS directive is 2048 characters.

3.3.3.6 The User's Tasks

These can be programs or commands. This is where the user specifies an application to be run.

3.3.3.7 Setting Job Attributes

Job attributes can be set by either of the following methods:

- Using PBS directives in the job script
- Giving options to the `qsub` command at the command line

These two methods have the same functionality. Options to the qsub command will override PBS directives, which override defaults. Some job attributes have default values preset in PBS. Some job attributes' default values are set at the user's site.

After the job is submitted, you can use the `qalter` command to change the job's characteristics.

Job attributes are case-insensitive.

3.3.3.8 Debugging Job Scripts

You can run Python interactively, outside of PBS, to debug a Python job script. You use the Python interpreter to test parts of your script.

Under UNIX/Linux, use the -i option to the `pbs_python` command, for example:

```
/opt/pbs/default/bin/pbs_python -i <return>
```

Under Windows, the -i option is not necessary, but can be used. For example, either of the following will work:

```
C:\Program Files\PBS Pro\exec\bin\pbs_python.exe
   <return>
C:\Program Files\PBS Pro\exec\bin\pbs_python.exe -i
   <return>
```

When the Python interpreter runs, it presents you with its own prompt. For example:

```
% /opt/pbs/default/bin/pbs_python -i <return>
>> print "hello"
hello
```

3.3.4 Job Script Names

It is recommended to avoid using special characters in a job script name. If you must use them, on UNIX/Linux you must escape them using the back-slash ("\") character.

3.4 Submitting a PBS Job

There are a few ways to submit a PBS job using the command line. The first is to create a job script and submit it using qsub.

3.4.1 Submitting a Job Script

For example, with job script "myjob", the user can submit it by typing

```
qsub myjob
16387.foo.exampledomain
```

PBS returns a *job identifier* (e.g. "`16387.foo.exampledomain`" in the example above.) Its format will be:

```
sequence-number.servername
```

or, for a job array,

```
sequence-number[].servername.domain
```

You'll need the job identifier for any actions involving the job, such as checking job status, modifying the job, tracking the job, or deleting the job.

If "my_job" contains the following, the user is naming the job "testjob", and running a program called "myprogram".

```
#!/bin/sh
#PBS -N testjob
./myprogram
```

The largest possible job ID is the 7-digit number 9,999,999. After this has been reached, job IDs start again at zero.

3.4.1.1 Overriding Directives

PBS directives in a script can be overridden by using the equivalent options to qsub. For example, to override the PBS directive naming the job, and name it "newjob", the user could type

```
qsub -N newjob my_job
```

3.4.1.2 Submitting a Simple Job

Jobs can also be submitted without specifying values for attributes. The simplest way to submit a job is to type

```
qsub myjobscript <ret>
```

If `myjobscript` contains

```
#!/bin/sh
./myapplication
```

the user has simply told PBS to run myapplication.

3.4.1.3 Passing Arguments to Job Scripts

If you need to pass arguments to a job script, you can either use the -v option to qsub, where you set and use environment variables, or use standard input. When using standard input, any #PBS directives in the job script will be ignored. You can replace directives with the equivalent options to qsub. To use standard input, you can either use this form:

```
echo "jobscript.sh -a foo -b bar" | qsub -l select=...
```

or you can use this form:

```
qsub [option] [option] ... <ret>
./jobscript.sh foo      <^d>
152.mymachine
```

With this form, you can type the #PBS directives on lines the name of the job script. If you do not use the -n option to qsub, or specify it via a #PBS directive (second form only), the job will be named STDIN.

3.4.2 Jobs Without a Job Script

There are two ways to submit PBS jobs without using a job script. You can run a PBS job by specifying an executable and its arguments instead of a job script. You can also specify that qsub read input from the keyboard.

3.4.2.1 Submitting Jobs by Specifying Executables

When you specify only the executable with any options and arguments, PBS starts a shell for you. To submit a job from the command line, the format is the following:

qsub [options] -- executable [arguments to executable] <return>

For example, to run `myprog` with the arguments `a` and `b`:

```
qsub -- myprog a b <return>
```

To run `myprog` with the arguments `a` and `b`, naming the job *JobA*,

```
qsub -N JobA -- myprog a b <return>
```

3.4.2.2 Submitting Jobs Using Keyboard Input

It is possible to submit a job to PBS without first creating a job script file. If you run the `qsub` command, with the resource requests on the command line, and then press "enter" without naming a job file, PBS will read input from the keyboard. (This is often referred to as a "here document".) You can direct `qsub` to stop reading input and submit the job by typing on a line by itself a `control-d` (UNIX) or `control-z`, then `enter` (Windows).

Note that, under UNIX, if you enter a `control-c` while `qsub` is reading input, `qsub` will terminate the process and the job will not be submitted. Under Windows, however, often the `control-c` sequence will, depend-

ing on the command prompt used, cause `qsub` to submit the job to PBS. In such case, a `control-break` sequence will usually terminate the `qsub` command.

```
qsub <ret>
    [directives]
    [tasks]
    ctrl-D
```

3.5 Requesting Resources

PBS provides built-in resources, and allows the administrator to define custom resources. The administrator can specify which resources are available on a given vnode, as well as at the queue or server level (e.g. floating licenses.) See “Built-in Resources” on page 350 of the PBS Professional Reference Guide for a listing of built-in resources.

Resources defined at the queue or server level apply to an entire job. If they are defined at the vnode level, they apply only to the part of the job running on that vnode.

Jobs request resources, which are allocated to the job, along with any defaults specified by the administrator.

Custom resources are used for application licenses, scratch space, etc., and are defined by the administrator. See section 5.14, "Custom Resources", on page 352 of the PBS Professional Administrator’s Guide of the PBS Professional Administrator’s Guide. Custom resources are used the same way built-in resources are used.

Jobs request resources in two ways. They can use the *select statement* to define *chunks* and specify the quantity of each chunk. A chunk is a set of resources that are to be allocated as a unit. Jobs can also use a job-wide resource request, which uses `resource=value` pairs, outside of the select statement. Format:

```
qsub (non-resource portion of job)
-l <resource>=<value>            (this is the job-wide request)
-l select=<chunk>[+<chunk>]      (this is the selection statement)
```

See section 3.5.2 “Requesting Resources in Chunks” on page 38 and section 3.5.3 “Requesting Job-wide Resources” on page 39.

The `qsub`, `qalter` and `pbs_rsub` commands are used to request resources. However, custom resources which were created to be invisible or unrequestable cannot be requested. See section 3.5.15 “Resource Permissions” on page 49.

The `-lnodes=` form is deprecated, and if it is used, it will be converted into a request for chunks and job-wide resources. Most jobs submitted with "-lnodes" will continue to work as expected. These jobs will be automatically converted to the new syntax. However, job tasks may execute in an unexpected order, because vnodes may be assigned in a different order. Jobs submitted with old syntax that ran successfully on versions of PBS Professional prior to 8.0 can fail because a limit that was per-chunk is now job-wide. This is an example of a job submitted using -lnodes=X -lmem=M that would fail because the mem limit is now job-wide. If the following conditions are true:

a. PBS Professional 9.0 or later using standard MPICH

b. The job is submitted with qsub -lnodes=5 -lmem=10GB

c. The master process of this job tries to use more than 2GB

The job will be killed, where in <= 7.0 the master process could use 10GB before being killed. 10GB is now a job-wide limit, divided up into a 2GB limit per chunk.

For more information see the `qsub(1B)`, `qalter(1B)`, `pbs_rsub(1B)` and `pbs_resources(7B)` manual pages.

Do not use an old-style resource or node specification (“-lnodes=”) with “-lselect” or “-lplace”. This will produce an error.

Each kind of resource plays a specific role, which is either inside chunks or outside of them, but not both. Some resources, e.g. ncpus, can only be used at the host (chunk) level. The rest, e.g. walltime, can only be used at the job-wide level. Therefore, no resource can be requested both inside and outside of a selection statement. Keep in mind that requesting, for example, -lncpus is the old form, which cannot be mixed with the new form.

3.5.1 Resource Allocation

Resources are allocated to jobs both because jobs explicitly request them and because specified default resources are applied to jobs. Jobs explicitly request resources either at the vnode level in *chunks* defined in a *selection statement*, or in *job-wide* resource requests, outside of a selection statement. An explicit resource request can appear in the following, in order of precedence:

1. qalter
2. qsub
3. PBS job script directives

3.5.2 Requesting Resources in Chunks

A *chunk* declares the value of each resource in a set of resources which are to be allocated as a unit to a job. It is the smallest set of resources that will be allocated to a job. All of a chunk must be taken from a single host. A chunk request is a vnode-level request. Chunks are described in a selection statement, which specifies how many of each kind of chunk. A selection statement has this form:

-l select=[N:]chunk[+[N:]chunk ...]

If N is not specified, it is taken to be 1.

No spaces are allowed between chunks.

A chunk is one or more resource_name=value statements separated by a colon, e.g.:

```
ncpus=2:mem=10GB:host=Host1
ncpus=1:mem=20GB:arch=linux
```

Example of multiple chunks in a selection statement:

```
-l select= 2:ncpus=1:mem=10GB
   +3:ncpus=2:mem=8GB:arch=solaris
```

Each job submission can have only one “-l select” statement.

Host-level resources can only be requested as part of a chunk. Server or queue resources cannot be requested as part of a chunk. They must be requested outside of the selection statement.

3.5.3 Requesting Job-wide Resources

A *job-wide* resource request is for resource(s) at the server or queue level. This resource must be a server-level or queue-level resource. A job-wide resource is designed to be used by the entire job, and is available to the complex, not just one execution host.

Job-wide resources are requested outside of a selection statement, in this form:

-l keyword=value[,keyword=value ...]

where *keyword* identifies either a consumable resource or a time-based resource such as walltime.

Job-wide resources are used for requesting floating licenses or other resources not tied to specific vnodes, such as cput and walltime.

Job-wide resources can only be requested outside of chunks.

A resource request "outside of a selection statement" means that the resource request comes after "`-l`", but not after "`-lselect=`".

3.5.4 Boolean Resources

A resource request can specify whether a boolean resource should be true or false. For example, if some vnodes have green=true and some are red=true, a selection statement for two vnodes, each with one CPU, all green and no red, would be:

```
-l select=2:green=true:red=false:ncpus=1
```

The next example Windows script shows a job-wide request for walltime and a chunk request for ncpus and memory.

```
#PBS -l walltime=1:00:00
#PBS -l select=ncpus=4:mem=400mb
#PBS -j oe

date /t
.\my_application
date /t
```

Keep in mind the difference between requesting a vnode-level boolean and a job-wide boolean.

```
qsub -l select=1:green=True
```

will request a vnode with green set to True. However,

```
qsub -l green=True
```

will request green set to True on the server and/or queue.

3.5.5 Default Resources

Jobs get default resources, both job-wide and per-chunk, with the following order of precedence, from

1. Default `qsub` arguments
2. Default queue resources
3. Default server resources

For each chunk in the job's selection statement, first queue chunk defaults are applied, then server chunk defaults are applied. If the chunk request does not specify a resource listed in the defaults, the default is added. For a resource RESOURCE, a chunk default is called "default_chunk.RESOURCE".

For example, if the queue in which the job is enqueued has the following defaults defined:

```
default_chunk.ncpus=1
default_chunk.mem=2gb
```

a job submitted with this selection statement:

```
select=2:ncpus=4+1:mem=9gb
```

will have this specification after the default_chunk elements are applied:

```
select=2:ncpus=4:mem=2gb+1:ncpus=1:mem=9gb.
```

In the above, *mem=2gb* and *ncpus=1* are inherited from default_chunk.

The job-wide resource request is checked against queue resource defaults, then against server resource defaults. If a default resource is defined which is not specified in the resource request, it is added to the resource request.

3.5.6 Requesting Application Licenses

Application licenses are set up as resources defined by the administrator. PBS doesn't actually check out the licenses, the application being run inside the job's session does that.

3.5.6.1 Floating Licenses

PBS queries the license server to find out how many floating licenses are available at the beginning of each scheduling cycle. If you wish to request a site-wide floating license, it will typically have been set up as a server-level (job-wide) resource. To request an application license called AppF, use:

```
qsub -l AppF=<number of licenses> <other qsub
        arguments>
```

If only certain hosts can run the application, they will typically have a host-level boolean resource set to True. To request the application license and the vnodes on which to run the application, use:

```
qsub  -l AppF=<number of licenses>
         <other qsub arguments>
      -l select=haveAppF=True
```

PBS doesn't actually check out the licenses, the application being run inside the job's session does that.

3.5.6.2 Node-locked Licenses

Per-host node-locked licenses are typically set up as either a boolean resource on the vnode(s) that are licensed for the application. The resource request should include one license for each host. To request a host with a per-host node-locked license for AppA in one chunk:

```
qsub -l select=1:runsAppA=1 <jobscript>
```

Per-use node-locked licenses are typically set up so that the host(s) that run the application have the number of licenses that can be used at one time. The number of licenses the job requests should be the same as the number of instances of the application that will be run. To request a host with a per-use node-locked license for AppB, where you'll run one instance of AppB on two CPUs in one chunk:

```
qsub -l select=1:ncpus=2:AppB=1
```

Per-CPU node-locked licenses are set up so that the host has one license for each licensed CPU. You must request one license for each CPU. To request a host with a node-locked license for AppC, where you'll run a job using two CPUs in one chunk:

```
qsub -l select=1:ncpus=2:AppC=2
```

3.5.7 Requesting Scratch Space

Scratch space on a machine is set up as a host-level dynamic resource. The resource will have a name such as "dynscratch". To request 10MB of scratch space in one chunk, a resource request would include:

```
-l select=1:ncpus=N:dynscratch=10MB
```

3.5.8 Requesting GPUs

Your PBS job can request GPUs. Your administrator can configure PBS to support any of the following:

- Job uses non-specific GPUs and exclusive use of a node
- Job uses non-specific GPUs and shared use of a node
- Job uses specific GPUs and either shared or exclusive use of a node

3.5.8.1 Binding to GPUs

PBS Professional allocates GPUs, but does not bind jobs to any particular GPU; the application itself, or the CUDA library, is responsible for the actual binding.

3.5.8.2 Requesting Non-specific GPUs and Exclusive Use of Node

If your job needs GPUs, but does not require specific GPUs, and can request exclusive use of GPU nodes, you can request GPUs the same way you request CPUs.

Your administrator can set up a resource to represent the GPUs on a node. We recommend that the GPU resource is called *ngpus*.

You submit jobs, for example "*my_gpu_job*", requesting one node with two GPUs and one CPU, and exclusive use of the node, in the following manner:

```
qsub -lselect=1:ncpus=1:ngpus=2 -lplace=excl my_gpu_job
```

When requesting GPUs in this manner, your job should request exclusive use of the node to prevent other jobs being scheduled on its GPUs.

It is up to the application or CUDA to bind the GPUs to the application processes.

3.5.8.3 Requesting Non-specific GPUs and Shared Use of Node

Your administrator can configure PBS to allow your job to use non-specific GPUs on a node while sharing GPU nodes. In this case, your administrator puts each GPU in its own vnode.

Your administrator can configure a resource to represent GPUs. We recommend that the GPU resource is called *ngpus*.

Your administrator can configure each GPU vnode so it has a resource containing the device number of the GPU. We recommend that this resource is called *gpu_id*.

You submit jobs, for example "*my_gpu_job*", requesting two GPUs and one CPU, and shared use of the node, in the following manner:

```
qsub -lselect=1:ncpus=1:ngpus=2 -lplace=shared
   my_gpu_job
```

When a job is submitted requesting any GPU, the PBS scheduler looks for a vnode with an available GPU and assigns that vnode to the job. Since there is a one-to-one correspondence between GPUs and vnodes, the job can determine the gpu_id of that vnode. Finally, the application can use the appropriate CUDA call to bind the process to the allocated GPU.

3.5.8.4 Requesting Specific GPUs

Your job can request one or more particular GPUs. This allows you to run applications on the GPUs for which the applications are written.

Your administrator can set up a resource to allow jobs to request specific GPUs. We recommend that the GPU resource is called *gpu_id*.

The following requests 4 vnodes, each with GPU with ID *0*:

```
qsub -lselect=4:ncpus=1:gpu_id=gpu0 my_gpu_job
```

When a job is submitted requesting specific GPUs, the PBS scheduler assigns the vnode with the resource containing that gpu_id to the job. The application can use the appropriate CUDA call to bind the process to the allocated GPU.

3.5.8.5 Viewing GPU Information for Nodes

You can find the number of GPUs available and assigned on execution hosts via the `pbsnodes` command. See "pbsnodes" on page 123 of the PBS Professional Reference Guide.

3.5.9 Note About Submitting Jobs

The default for walltime is 5 years. The scheduler uses walltime to predict when resources will become available. Therefore it is useful to request a reasonable walltime for each job.

3.5.10 Submitting Jobs with Resource Specification (Old Syntax)

If neither a node specification nor a selection directive is specified, then a selection directive will be created requesting 1 chunk with resources specified by the job, and with those from the queue or server default resource list. These are: ncpus, mem, arch, host, and software, as well as any other default resources specified by the administrator.

For example, a job submitted with

```
qsub -l ncpus=4:mem=123mb:arch=linux
```

will have the following selection directive created:

```
select=1:ncpus=4:mem=123mb:arch=linux
```

Do not mix old style resource or node specification with the select and place statements. Do not use one in a job script and the other on the command line. This will result in an error.

3.5.11 Moving Jobs From One Queue to Another

If the job is moved from the current queue to a new queue, any default resources in the job's resource list that were contributed by the current queue are removed. This includes a select specification and place directive generated by the rules for conversion from the old syntax. If a job's resource is unset (undefined) and there exists a default value at the new queue or server, that default value is applied to the job's resource list. If either select or place is missing from the job's new resource list, it will be automatically generated, using any newly inherited default values.

Example:

Given the following set of queue and server default values:

Server

```
resources_default.ncpus=1
```

Queue QA

```
resources_default.ncpus=2
default_chunk.mem=2gb
```

Queue QB

```
default_chunk.mem=1gb
```

no default for ncpus

The following illustrate the equivalent select specification for jobs submitted into queue QA and then moved to (or submitted directly to) queue QB:

`qsub -l ncpus=1 -lmem=4gb`

In QA: `select=1:ncpus=1:mem=4gb`

No defaults need be applied

In QB: `select=1:ncpus=1:mem=4gb`

No defaults need be applied

qsub -l ncpus=1

In QA: `select=1:ncpus=1:mem=2gb`

Picks up 2gb from queue default chunk and 1 ncpus from qsub

In QB: `select=1:ncpus=1:mem=1gb`

Picks up 1gb from queue default chunk and 1 ncpus from qsub

qsub -lmem=4gb

In QA: `select=1:ncpus=2:mem=4gb`

Picks up 2 ncpus from queue level job-wide resource default and 4gb mem from qsub

In QB: `select=1:ncpus=1:mem=4gb`

Picks up 1 ncpus from server level job-wide default and 4gb mem from qsub

qsub -lnodes=4

In QA: `select=4:ncpus=1:mem=2gb`

Picks up a queue level default memory chunk of 2gb. (This is not 4:ncpus=2 because in prior versions, "nodes=x" implied 1 CPU per node unless otherwise explicitly stated.)

In QB: `select=4:ncpus=1:mem=1gb` (In prior versions, "nodes=x" implied 1 CPU per node unless otherwise explicitly stated, so the ncpus=1 is not inherited from the server default.)

`qsub -l mem=16gb -lnodes=4`

In QA: `select=4:ncpus=1:mem=4gb` (This is not 4:ncpus=2 because in prior versions, "nodes=x" implied 1 CPU per node unless otherwise explicitly stated.)

In QB: `select=4:ncpus=1:mem=4gb` (In prior versions, "nodes=x" implied 1 CPU per node unless otherwise explicitly stated, so the ncpus=1 is not inherited from the server default.)

3.5.12 Resource Request Conversion Dependent on Where Resources are Defined

A job's resource request is converted from old-style to new according to various rules, one of which is that the conversion is dependent upon where resources are defined. For example: The boolean resource "Red" is defined on the server, and the boolean resource "Blue" is defined at the host level. A job requests "qsub -l Blue=True". This looks like an old-style resource request, and PBS checks to see where Blue is defined. Since Blue is defined at the host level, the request is converted into "-l select=1:Blue=True". However, if a job requests "qsub -l Red=True", while this looks like an old-style resource request, PBS does not convert it to a chunk request because Red is defined at the server.

3.5.13 Jobs Submitted with Undefined Resources

Any job submitted with undefined resources, specified either with "-l select" or with "-lnodes", will not be rejected at submission. The job will be aborted upon being enqueued in an execution queue if the resources are still undefined. This preserves backward compatibility.

3.5.14 Limits on Resource Usage

Each chunk's per-chunk limits determine how much of any resource can be used in that chunk. Per-chunk resource usage limits are established by per-chunk resources, both from explicit requests and from defaults.

Job resource limits set a limit for per-job resource usage. Job resource limits are established both by requesting job-wide resources and by summing per-chunk consumable resources. Job resource limits from sums of all chunks, including defaults, override those from job-wide defaults. Limits include both explicitly requested resources and default resources.

If a job's job resource limit exceeds queue or server restrictions, it will not be put in the queue or accepted by the server. If, while running, a job exceeds its limit for a consumable or time-based resource, it will be terminated. See The **PBS Professional Administrator's Guide**.

Job limits are created from the directive for each consumable resource.

For example,

```
qsub -lselect=2:ncpus=3:mem=4gb:arch=linux
```

will have the following job limits set:

- ncpus=6
- mem=8gb

3.5.15 Resource Permissions

Custom resources can be created so that they are invisible, or cannot be requested or altered. If a resource is invisible it also cannot be requested or altered. The function of some PBS commands depends upon whether a resource can be viewed, requested or altered. These commands are those which view or request resources or modify resource requests:

pbsnodes
: Users cannot view restricted host-level custom resources.

pbs_rstat
: Users cannot view restricted reservation resources.

pbs_rsub
: Users cannot request restricted custom resources for reservations.

qalter
: Users cannot alter a restricted resource.

qmgr
: Users cannot print or list a restricted resource.

`qselect`
: Users cannot specify restricted resources via -l resource_list.

qsub
: Users cannot request a restricted resource.

qstat
: Users cannot view a restricted resource.

3.6 Placing Jobs on Vnodes

The *place statement* and the vnode sharing attribute controls how the job is placed on the vnodes from which resources may be allocated for the job. The place statement can be specified, in order of precedence, via:

1. Explicit placement request in qalter
2. Explicit placement request in qsub
3. Explicit placement request in PBS job script directives
4. Default `qsub` place statement
5. Queue default placement rules
6. Server default placement rules
7. Built-in default conversion and placement rules

The place statement may be not be used without the select statement.

The place statement has this form:

-l place=[arrangement][: sharing][: grouping]

where

arrangement is one of `free | pack | scatter | vscatter`

sharing is one of `excl | shared | exclhost`

grouping can have only one instance of `group=resource`

and where

Table 3-1: Placement Modifiers

Modifier	Meaning
free	Place job on any vnode(s)
pack	All chunks will be taken from one host
scatter	Only one chunk is taken from any host
vscatter	Only one chunk is taken from any vnode. Each chunk must fit on a vnode.
excl	Only this job uses the vnodes chosen
exclhost	The entire host is allocated to the job
shared	This job can share the vnodes chosen
group=resource	Chunks will be placed on vnodes according to a resource shared by those vnodes. This resource must be a string or string array. All vnodes in the group must have a common value for the resource, which can be either the built-in resource host or a site-defined vnode-level resource

Grouping by resource name will override `node_group_key`. To run a job on a single host, use "-lplace=pack".

3.6.1 Inheriting Placement Specifications

If the administrator has defined default values for arrangement, sharing, and grouping, each job inherits these unless it explicitly requests at least one. That means that if your job requests arrangement, but not sharing or

grouping, it will not inherit values for sharing or grouping. For example, the administrator sets a default of `place=pack:exclhost:group=host`. Job A requests `place=free`, but doesn't specify sharing or grouping, so Job A does not inherit sharing or grouping. Job B does not request any placement, so it inherits all three.

3.6.2 Using *place=group*

When a job requests `place=group=<resource>`, PBS looks for enough vnodes to satisfy the request, where all of the selected vnodes share a common value for the specified resource.

The specified resource must be a string or string array.

The value of the resource can be one or more strings at each vnode, but there must be one string that is the same for each vnode. For example, if the resource is *router*, the value can be "*r1i0,r1*" at one vnode, and "*r1i1,r1*" at another vnode, but these vnodes can be grouped because they share the string "*r1*".

3.6.3 Specifying Primary Execution Host

The job's primary execution host is the host that supplies the vnode to satisfy first chunk requested by the job. You can see which host is the primary execution host in the following ways:

- The primary execution host is the first host listed in the job's nodes file; see section 3.6.5 "The PBS Node File" on page 55.
- The primary execution host is the first host listed in the job's exec_vnode attribute; see section 6.1.19 "Viewing Resources Allocated to a Job" on page 148.

3.6.4 Vnodes Allocated to a Job

3.6.4.1 Hosts Allocated to a Job

The nodes file contains the names of the vnodes allocated to a job. The nodes file's name is given by the environment variable PBS_NODEFILE. The order in which hosts appear in the file is the order in which chunks are specified in the selection directive. The order in which hostnames appear in the file is hostA X times, hostB Y times, where X is the number of MPI processes on hostA, Y is the number of MPI processes on hostB, etc. See section 3.6.5 "The PBS Node File" on page 55. See also the definition of the resources "mpiprocs" and "ompthreads" in "Built-in Resources" on page 350 of the PBS Professional Reference Guide, and "The mpiprocs Resource" on page 231.

3.6.4.2 Specifying Shared or Exclusive Use of Vnodes

Each vnode can be allocated exclusively to one job, or its resources can be shared among jobs. Hosts can also be allocated exclusively to one job, or shared among jobs.

How vnodes are allocated to jobs is determined by a combination of the vnode's sharing attribute and the job's resource request. The possible values for the vnode sharing attribute, and how they interact with a job's placement request, are described in "sharing" on page 426 of the PBS Professional Reference Guide. The following table expands on this:

Table 3-2: How Vnode sharing Attribute Affects Vnode Allocation

Value of Vnode sharing Attribute	Effect on Allocation
not set	The job's arrangement request determines how vnodes are allocated to the job. If there is no specification, vnodes are shared.
default_share	Vnodes are shared unless the job explicitly requests exclusive use of the vnodes.
default_excl	Vnodes are allocated exclusively to the job unless the job explicitly requests shared allocation.

Table 3-2: How Vnode sharing Attribute Affects Vnode Allocation

Value of Vnode sharing Attribute	Effect on Allocation
default_exclhost	All vnodes from this host are allocated exclusively to the job, unless the job explicitly requests shared allocation.
ignore_excl	Vnodes are shared, regardless of the job's request.
force_excl	Vnodes are allocated exclusively to the job, regardless of the job's request.
force_exclhost	All vnodes from this host are allocated exclusively to the job, regardless of the job's request.

If a vnode is allocated exclusively to a job, all of its resources are assigned to the job. The state of the vnode becomes *job-exclusive*. No other job can use the vnode.

If a host is to be allocated exclusively to one job, all of the host must be used: if any vnode from a host has its sharing attribute set to either *default_exclhost* or *force_exclhost*, all vnodes on that host must have the same value for the sharing attribute.

3.6.4.3 Placing Jobs on Vnodes

Jobs can be placed on vnodes according to the job's placement request. Each chunk from a job can be placed on a different host, or a different vnode. Alternatively, all chunks can be taken from a single host, or from chunks sharing the same value for a specified resource. The job can request exclusive use of each vnode, or shared use with other jobs. The job can request exclusive use of its hosts. If a job does not request *vscatter* for its arrangement, any chunk can be broken across vnodes. This means that one chunk could be taken from more than one vnode.

3.6.4.4 Restrictions on Placement

If the job requests *vscatter* for its arrangement, no chunk can be larger than a vnode. No chunk can be split across vnodes. This behavior is different from other values for arrangement, where chunks can be split across vnodes.

3.6.5 The PBS Node File

The file containing the vnodes allocated to a job lists vnode names. This file's name is given by the environment variable PBS_NODEFILE. For jobs which request vnodes via the -lselect= option, the nodes file will contain the names of the allocated vnodes with each name repeated M times, where M is the number of mpiprocs specified for that vnode. For example,

```
qsub -l select=3:ncpus=2 -lplace=scatter
```

will result in this PBS_NODEFILE:

```
vnodeA
vnodeB
vnodeC
```

And

```
qsub -l select=3:ncpus=2:mpiprocs=2
```

will result in this PBS_NODEFILE:

```
vnodeA
vnodeA
vnodeB
vnodeB
vnodeC
vnodeC
```

For jobs which requested a set of nodes via the -lnodes=nodespec option to qsub, each vnode allocated to the job will be listed N times, where N is the total number of CPUs allocated from the vnode divided by the number of threads requested. For example, qsub -lnodes=4:ncpus=3:ppn=2 will result in each of the four vnodes being written twice (6 CPUs divided by 3 from

ncpus.) The file will contain the name of the first vnode twice, followed by the second vnode twice, etc. See section 12.12, "The Job's Node File", on page 862 of the PBS Professional Administrator's Guide.

3.6.6 Resources Allocated from a Vnode

The resources allocated from a vnode are only those specified in the job's *schedselect*. This job attribute is created internally by starting with the select specification and applying any server and queue default_chunk resource defaults that are missing from the select statement. The schedselect job attribute contains only vnode-level resources. The exec_vnode job attribute shows which resources are allocated from which vnodes. See "Job Attributes" on page 430 of the PBS Professional Reference Guide.

3.6.6.1 Resources Assigned to a Job

The Resource_List attribute is the list of resources requested via qsub, with job-wide defaults applied. Vnode-level resources from Resource_List are used in the converted select when the user doesn't specify a select statement. The converted select statement is used to fill in gaps in schedselect.

Values for ncpus or mem in the job's Resource_List come from three places:

1. Resources specified via qsub,
2. the sum of the values in the select specification (not including default_chunk), or
3. resources inherited from queue and/or server resources_default.

Case 3 applies only when the user does not specify -l select, but uses -lnodes or -lncpus instead.

The Resource_List.mem is a job-wide memory limit which, if memory enforcement is enabled, the entire job (the sum of all of the job's usage) cannot exceed.

Examples:

The queue has the following:

```
resources_default.mem=200mb
default_chunk.mem=100mb
```

A job requesting -l select=2:ncpus=1:mem=345mb will take 345mb from each of two vnodes and have a job-wide limit of 690mb (2 * 345). The job's Resource_List.mem will show 690mb.

A job requesting -l select=2:ncpus=2 will take 100mb (default_chunk) value from each vnode and have a job wide limit of 200mb (2 * 100mb). The job's Resource_List.mem will show 200mb.

A job requesting -l ncpus=2 will take 200mb (inherited from resources_default and used to create the select spec) from one vnode and a job-wide limit of 200mb. The job's Resource_List.mem will show 200mb.

A job requesting -lnodes=2 will inherit the 200mb from resources_default.mem which will be the job-wide limit. The memory will be taken from the two vnodes, half (100mb) from each. The generated select spec is 2:ncpus=1:mem=100mb. The job's Resource_List.mem will show 200mb.

3.7 Submitting Jobs Using Select & Place: Examples

Unless otherwise specified, the vnodes allocated to the job will be allocated as shared or exclusive based on the setting of the vnode's sharing attribute. Each of the following shows how you would use -l select= and -l place=.

1. A job that will fit in a single host such as an Altix but not in any of the vnodes, packed into the fewest vnodes:

   ```
   -l select=1:ncpus=10:mem=20gb
   -l place=pack
   ```

   In earlier versions, this would have been:

   ```
   -lncpus=10,mem=20gb
   ```

2. Request four chunks, each with 1 CPU and 4GB of memory taken from anywhere.

   ```
   -l select=4:ncpus=1:mem=4GB
   -l place=free
   ```

3. Allocate 4 chunks, each with 1 CPU and 2GB of memory from between

one and four vnodes which have an arch of “linux”.

```
-l select=4:ncpus=1:mem=2GB:arch=linux -l place=free
```

4. Allocate four chunks on 1 to 4 vnodes where each vnode must have 1 CPU, 3GB of memory and 1 node-locked dyna license available for each chunk.

```
-l select=4:dyna=1:ncpus=1:mem=3GB -l place=free
```

5. Allocate four chunks on 1 to 4 vnodes, and 4 floating dyna licenses. This assumes “dyna” is specified as a server dynamic resource.

```
-l dyna=4 -l select=4:ncpus=1:mem=3GB -l place=free
```

6. This selects exactly 4 vnodes where the arch is linux, and each vnode will be on a separate host. Each vnode will have 1 CPU and 2GB of memory allocated to the job.

```
-lselect=4:mem=2GB:ncpus=1:arch=linux -lplace=scatter
```

7. This will allocate 3 chunks, each with 1 CPU and 10GB of memory. This will also reserve 100mb of scratch space if scratch is to be accounted . Scratch is assumed to be on a file system common to all

hosts. The value of "place" depends on the default which is "place=free".

```
-l scratch=100mb -l select=3:ncpus=1:mem=10GB
```

8. This will allocate 2 CPUs and 50GB of memory on a host named zooland. The value of "place" depends on the default which defaults to "place=free":

```
-l select=1:ncpus=2:mem=50gb:host=zooland
```

9. This will allocate 1 CPU and 6GB of memory and one host-locked swlicense from each of two hosts:

```
-l select=2:ncpus=1:mem=6gb:swlicense=1
-lplace=scatter
```

10. Request free placement of 10 CPUs across hosts:

```
-l select=10:ncpus=1
-l place=free
```

11. Here is an odd-sized job that will fit on a single Altix, but not on any one node-board. We request an odd number of CPUs that are not shared, so they must be "rounded up":

```
-l select=1:ncpus=3:mem=6gb
-l place=pack:excl
```

12. Here is an odd-sized job that will fit on a single Altix, but not on any one node-board. We are asking for small number of CPUs but a large amount of memory:

```
-l select=1:ncpus=1:mem=25gb
-l place=pack:excl
```

13. Here is a job that may be run across multiple Altix systems, packed into the fewest vnodes:

```
-l select=2:ncpus=10:mem=12gb
-l place=free
```

14. Submit a job that must be run across multiple Altix systems, packed into the fewest vnodes:

```
-l select=2:ncpus=10:mem=12gb
```

```
-l place=scatter
```

15. Request free placement across nodeboards within a single host:

```
-l select=1:ncpus=10:mem=10gb
-l place=group=host
```

16. Request free placement across vnodes on multiple Altixes:

```
-l select=10:ncpus=1:mem=1gb
-l place=free
```

17. Here is a small job that uses a shared cpuset:

```
-l select=1:ncpus=1:mem=512kb
-l place=pack:shared
```

18. Request a special resource available on a limited set of nodeboards, such as a graphics card:

```
-l select= 1:ncpus=2:mem=2gb:graphics=True +
   1:ncpus=20:mem=20gb:graphics=False
-l place=pack:excl
```

19. Align SMP jobs on c-brick boundaries:

```
-l select=1:ncpus=4:mem=6gb
-l place=pack:group=cbrick
```

20. Align a large job within one router, if it fits within a router:

```
-l select=1:ncpus=100:mem=200gb
-l place=pack:group=router
```

21. Fit large jobs that do not fit within a single router into as few available routers as possible. Here, RES is the resource used for node grouping:

```
-l select=1:ncpus=300:mem=300gb
-l place=pack:group=<RES>
```

22. To submit an MPI job, specify one chunk per MPI task. For a 10-way MPI job with 2gb of memory per MPI task:

```
-l select=10:ncpus=1:mem=2gb
```

23. To submit a non-MPI job (including a 1-CPU job or an OpenMP or

shared memory) job, use a single chunk. For a 2-CPU job requiring 10gb of memory:

```
-l  select=1:ncpus=2:mem=10gb
```

3.7.1 Examples Using Old Syntax

1. Request CPUs and memory on a single host using old syntax:

   ```
   -l ncpus=5,mem=10gb
   ```

   will be converted into the equivalent:

   ```
   -l select=1:ncpus=5:mem=10gb
   -l place=pack
   ```

2. Request CPUs and memory on a named host along with custom resources including a floating license using old syntax:

   ```
   -l ncpus=1,mem=5mb,host=sunny,opti=1,arch=solaris
   ```

   is converted to the equivalent:

   ```
   -l select=1:ncpus=1:mem=5gb:host=sunny:arch=solaris
   -l place=pack
   -l opti=1
   ```

3. Request one host with a certain property using old syntax:

   ```
   -lnodes=1:property
   ```

   is converted to the equivalent:

   ```
   -l select=1:ncpus=1:property=True
   -l place=scatter
   ```

4. Request 2 CPUs on each of four hosts with a given property using old syntax:

   ```
   -lnodes=4:property:ncpus=2
   ```

   is converted to the equivalent:

   ```
   -l select=4: ncpus=2:property=True
   ```

-l place=scatter

5. Request 1 CPU on each of 14 hosts asking for certain software, licenses and a job limit amount of memory using old syntax:

```
-lnodes=14:mpi-fluent:ncpus=1 -lfluent=1,fluent-all=1,
   fluent-par=13
-l mem=280mb
```

is converted to the equivalent:

```
-l select=14:ncpus=1:mem=20mb:mpi_fluent=True
-l place=scatter
-l fluent=1,fluent-all=1,fluent-par=13
```

6. Requesting licenses using old syntax:

```
-lnodes=3:dyna-mpi-Linux:ncpus=2 -ldyna=6,mem=100mb,
   software=dyna
```

is converted to the equivalent:

```
-l select=3:ncpus=2:mem=33mb: dyna-mpi-Linux=True
-l place=scatter
-l software=dyna
-l dyna=6
```

7. Requesting licenses using old syntax:

```
 -l ncpus=2,app_lic=6,mem=200mb -l software=app
```

is converted to the equivalent:

```
-l select=1:ncpus=2:mem=200mb
-l place=pack
-l software=app
-l app_lic=6
```

8. Additional example using old syntax:

```
-lnodes=1:fserver+15:noserver
```

is converted to the equivalent:

```
-l select=1:ncpus=1:fserver=True +
   15:ncpus=1:noserver=True
```

```
-l place=scatter
```

but could also be more easily specified with something like:

```
-l select=1:ncpus=1:fserver=True +
   15:ncpus=1:fserver=False
-l place=scatter
```

9. Allocate 4 vnodes, each with 6 CPUs with 3 MPI processes per vnode, with each vnode on a separate host. The memory allocated would be one-fourth of the memory specified by the queue or server default if one existed. This results in a different placement of the job from version 5.4:

```
-lnodes=4:ppn=3:ncpus=2
```

is converted to:

```
-l select=4:ncpus=6:mpiprocs=3 -l place=scatter
```

10. Allocate 4 vnodes, from 4 separate hosts, with the property blue. The amount of memory allocated from each vnode is 2560MB (= 10GB / 4) rather than 10GB from each vnode.

```
-lnodes=4:blue:ncpus=2 -l mem=10GB
```

is converted to:

```
-l select=4:blue=True:ncpus=2:mem=2560mb -lplace=scat-
   ter
```

3.8 Backward Compatibility

For backward compatibility, a legal node specification or resource specification will be converted into selection and placement directives. Specifying “cpp” is part of the old syntax, and should be replaced with “ncpus”. Do not mix old style resource or node specification syntax with select and place statements. If a job is submitted using -l select on the command line, and it contains an old-style specification in the job script, that will result in an error.

When a nodespec is converted into a select statement, the job will have the environment variables NCPUS and OMP_NUM_THREADS set to the value of ncpus in the first piece of the nodespec. This may produce incompatibilities with prior versions when a complex node specification using different values of ncpus and ppn in different pieces is converted.

3.8.1 Node Specification Conversion

Node specification format:

-lnodes=[N:*spec_list* | *spec_list*]

[[+N:*spec_list* | +*spec_list*] ...]

[#*suffix* ...][-lncpus=Z]

where:

spec_list has syntax: *spec*[:*spec* ...]

spec is any of: hostname | property | ncpus=X | cpp=X | ppn=P

suffix is any of: property | excl | shared

N and P are positive integers

X and Z are non-negative integers

The node specification is converted into selection and placement directives as follows:

Each *spec_list* is converted into one chunk, so that N:*spec_list* is converted into N chunks.

If spec is hostname :

The chunk will include host=hostname

If spec matches any vnode's resources_available.host value:

The chunk will include host=hostname

If spec is property :

The chunk will include property=true

Property must be a site-defined vnode-level boolean resource.

If spec is ncpus=X or cpp=X :

The chunk will include ncpus=X

If no spec is ncpus=X and no spec is cpp=X :

The chunk will include ncpus=P

If spec is ppn=P :

The chunk will include mpiprocs=P

If the nodespec is

```
-lnodes=N:ppn=P
```

It is converted to

```
-lselect=N:ncpus=P:mpiprocs=P
```

Example:

```
-lnodes=4:ppn=2
```

is converted into

```
-lselect=4:ncpus=2:mpiprocs=2
```

If -lncpus=Z is specified and no spec contains ncpus=X and no spec is cpp=X :

Every chunk will include ncpus=W, where W is Z divided by the total number of chunks. (Note: W must be an integer; Z must be evenly divisible by the number of chunks.)

If property is a suffix :

All chunks will include property=true

If excl is a suffix :

The placement directive will be -lplace=scatter:excl

If shared is a suffix :

The placement directive will be -lplace=scatter:shared

If neither excl nor shared is a suffix :

The placement directive will be -lplace=scatter

Example:

```
-lnodes=3:green:ncpus=2:ppn=2+2:red
```

is converted to:

```
-l select=3:green=true:ncpus=4:mpiprocs=2+
   2:red=true:ncpus=1
-l place=scatter
```

Node specification syntax for requesting properties is deprecated. The boolean resource syntax "property=true" is only accepted in a selection directive. It is erroneous to mix old and new syntax.

3.8.2 Resource Specification Conversion

The resource specification is converted to select and place statements after any defaults have been applied.

Resource specification format:

-lresource=value[:resource=value ...]

The resource specification is converted to:

-lselect=1[:*resource=value* ...]

-lplace=pack

with one instance of *resource=value* for each of the following vnode-level resources in the resource request:

built-in resources: ncpus | mem | vmem | arch | host

site-defined vnode-level resources

3.9 How PBS Parses a Job Script

The `qsub` command scans the lines of the script file for directives. Scanning will continue until the first executable line, that is, a line that is not blank, not a directive line, nor a line whose first non white space character is “#”. If directives occur on subsequent lines, they will be ignored.

A line in the script file will be processed as a directive to qsub if and only if the string of characters starting with the first non white space character on the line and of the same length as the directive prefix matches the directive prefix (i.e. "#PBS"). The remainder of the directive line consists of the options to qsub in the same syntax as they appear on the command line. The option character is to be preceded with the "-" character.

If an option is present in both a directive and on the command line, that option and its argument, if any, will be ignored in the directive. The command line takes precedence. If an option is present in a directive and not on the command line, that option and its argument, if any, will be taken from there.

3.10 A Sample PBS Jobs

The following is an example of a job script written in Python. This script calculates the 10th Fibonacci number.

```
% cat job.py
#PBS -l select=1:ncpus=3:mem=1gb
#PBS -N myjob
def fibo(n):
  global fibo
  if n < 2:
        return n
  else:
        return fibo(n - 1) + fibo(n - 2)
print ("fibo(10)=%d"   % fibo(10))
```

Note that this script contains PBS directives.

Let's look at an example PBS job in detail:

UNIX/Linux:

```
#!/bin/sh
#PBS -l walltime=1:00:00
#PBS -l select=mem=400mb
#PBS -j oe

date
./my_application
date
```

Windows:

```
#PBS -l walltime=1:00:00
#PBS -l select=mem=400mb
#PBS -j oe

date /t
my_application
date /t
```

On line one in the example above Windows does not show a shell directive. (The default on Windows is the batch command language.) Also note that it is possible under both Windows and UNIX to specify to PBS the scripting language to use to interpret the job script (see the "-S" option to `qsub` in section 3.13.9 "Specifying Scripting Language to Use" on page 83). The Windows script will be a .exe or .bat file.

Lines 2-8 of both files are almost identical. The primary differences will be in file and directory path specification (such as the use of drive letters and slash vs. backslash as the path separator).

Lines 2-4 are PBS directives. PBS reads down the shell script until it finds the first line that is not a valid PBS directive, then stops. It assumes the rest of the script is the list of commands or tasks that the user wishes to run. In this case, PBS sees lines 6-8 as being user commands.

The section "Job Submission Options" on page 74 describes how to use the `qsub` command to submit PBS jobs. Any option that you specify to the `qsub` command line (except "`-I`") can also be provided as a PBS directive inside the PBS script. PBS directives come in two types: resource requirements and attribute settings.

In our example above, lines 2-3 specify the "`-l`" resource list option, followed by a specific resource request. Specifically, lines 2-3 request 1 hour of wall-clock time as a job-wide request, and 400 megabytes (MB) of memory in a chunk. .

Line 4 requests that PBS *join* the `stdout` and `stderr` output streams of the job into a single stream.

Finally lines 6-8 are the command lines for executing the program(s) we wish to run. You can specify as many programs, tasks, or job steps as you need.

3.11 Changing the Job's PBS Directive

By default, the text string "`#PBS`" is used by PBS to determine which lines in the job file are PBS directives. The leading "#" symbol was chosen because it is a comment delimiter to all shell scripting languages in common use on UNIX systems. Because directives look like comments, the scripting language ignores them. The limit for a PBS directive is 2048 characters.

Under Windows, however, the command interpreter does not recognize the '#' symbol as a comment, and will generate a benign, non-fatal warning when it encounters each "`#PBS`" string. While it does not cause a problem for the batch job, it can be annoying or disconcerting to the user. Therefore Windows users may wish to specify a different PBS directive, via either the

PBS_DPREFIX environment variable, or the "-C" option to qsub. For example, we can direct PBS to use the string "REM PBS" instead of "#PBS" and use this directive string in our job script:

```
REM PBS -l walltime=1:00:00
REM PBS -l select=mem=400mb
REM PBS -j oe
date /t
.\my_application
date /t
```

Given the above job script, we can submit it to PBS in one of two ways:

```
set PBS_DPREFIX=REM PBS
qsub my_job_script
```

or

```
qsub -C "REM PBS" my_job_script
```

For additional details on the "-C" option to qsub, see section 3.13 "Job Submission Options" on page 74.

3.12 Windows Jobs

3.12.1 Submitting Windows Jobs

Any .bat files that are to be executed within a PBS job script have to be prefixed with "call" as in:

```
@echo off
call E:\step1.bat
call E:\step2.bat
```

Without the "call", only the first .bat file gets executed and it doesn't return control to the calling interpreter.

An example:

A job script that contains:

```
@echo off
E:\step1.bat
E:\step2.bat
```

should now be:

```
@echo off
call E:\step1.bat
call E:\step2.bat
```

Under Windows, comments in the job script must be in ASCII characters.

3.12.2 Passwords

When running PBS in a password-protected Windows environment, you will need to specify to PBS the password needed in order to run your jobs. There are two methods of doing this: (1) by providing PBS with a password once to be used for all jobs ("single signon method"), or (2) by specifying the password for each job when submitted ("per job method"). Check with your system administrator to see which method was configured at your site.

3.12.2.1 Single-Signon Password Method

To provide PBS with a password to be used for all your PBS jobs, use the `pbs_password` command. This command can be used whether or not you have jobs enqueued in PBS. The command usage syntax is:

pbs_password [-s server] [-r] [-d] [user]

When no options are given to pbs_password, the password credential on the default PBS server for the current user, i.e. the user who executes the command, is updated to the prompted password. Any user jobs previously held due to an invalid password are not released.

The available options to `pbs_password` are:

-r

Any user jobs previously held due to an invalid password are released.

-s server

Allows user to specify server where password will be changed.

-d

Deletes the password.

user

The password credential of user *user* is updated to the prompted password. If *user* is not the current user, this action is only allowed if:

1. The current user is root or admin.
2. User *user* has given the current user explicit access via the `ruserok()` mechanism:
   1 The hostname of the machine from which the current user is logged in appears in the server's hosts.equiv file, or
   2 The current user has an entry in user's `HOMEDIR\.rhosts` file.

Note that `pbs_password` encrypts the password obtained from the user before sending it to the PBS Server. The `pbs_password` command does not change the user's password on the current host, only the password that is cached in PBS.

The `pbs_password` command is supported only on Windows and all supported Linux platforms on x86 and x86_64.

The `pbs_password` command has no effect on running jobs. Queued jobs use the new password.

3.12.2.2 Per-job Password Method

If you are running in a password-protected Windows environment, but the single-signon method has not been configured at your site, then you will need to supply a password with the submission of each job. You can do this via the `qsub` command, with the -Wpwd option, and supply the password when prompted.

```
qsub -Wpwd <job script>
```

You will be prompted for the password , which is passed on to the program, then encrypted and saved securely for use by the job. The password should be enclosed in double quotes.

Keep in mind that in a multi-host job, the password supplied will be propagated to all the sister hosts. This requires that the password be the same on the user's accounts on all the hosts. The use of domain accounts for a multi-host job will be ideal in this case.

Accessing network share drives/resources within a job session also requires that you submit the job with a password via `qsub -W pwd`.

The -Wpwd option to the `qsub` command is supported only on Windows and all supported Linux platforms on x86 and x86_64.

3.13 Job Submission Options

There are many options to the `qsub` command. The table below gives a quick summary of the available options; the rest of this chapter explains how to use each one.

Table 3-3: Options to the `qsub` Command

Option	Function and Page Reference
`-A account_string`	“Specifying a Local Account” on page 90
`-a date_time`	“Deferring Execution” on page 84
`-C “DPREFIX”`	“Changing the Job’s PBS Directive” on page 70

Table 3-3: Options to the `qsub` Command

Option	Function and Page Reference
`-c interval`	“Specifying Job Checkpoint Interval” on page 86
`-e path`	“Specifying Path for Output and Error Files” on page 77
`-h`	“Holding a Job (Delaying Execution)” on page 85
`-I`	“Interactive-batch Jobs” on page 91
`-J X-Y[:Z]`	“Job Array” on page 205
`-j` *join*	“Merging Output and Error Files” on page 78
`-k` *keep*	“Retaining Output and Error Files on Execution Host” on page 79
`-l resource_list`	section 3.3.1 “Rules for Submitting Jobs” on page 27
`-M user_list`	“Setting Email Recipient List” on page 82
`-m MailOptions`	“Specifying Email Notification” on page 81
`-N name`	“Specifying a Job Name” on page 82
`-o path`	“Specifying Path for Output and Error Files” on page 77
`-p priority`	“Setting a Job’s Priority” on page 84
`-q destination`	“Specifying Queue and/or Server” on page 77
`-r value`	“Marking a Job as “Rerunnable” or Not” on page 83
`-S path_list`	“Specifying Scripting Language to Use” on page 83
`-u user_list`	“Specifying Job User ID” on page 87
`-V`	“Exporting Environment Variables” on page 80

Table 3-3: Options to the `qsub` Command

Option	Function and Page Reference
`-v variable_list`	"Expanding Environment Variables" on page 80
`-W <attribute>=<value>`	"Setting Job Attribute Values" on page 89
`-W depend=list`	"Specifying Job Dependencies" on page 161
`-W group_list=list`	"Specifying Job Group ID" on page 90
`-W stagein=list`	"Input/Output File Staging" on page 166
`-W stageout=list`	"Input/Output File Staging" on page 166
`-W cred=dce`	"Running PBS in a UNIX DCE Environment" on page 201
`-W block=opt`	"Requesting qsub Wait for Job Completion" on page 160
`-W pwd="password"`	"Per-job Password Method" on page 74 and "Running PBS in a UNIX DCE Environment" on page 201
`-W sandbox=<value>`	"Staging and Execution Directory: User's Home vs. Job-specific" on page 166
`-W umask=nnn`	"Changing UNIX Job umask" on page 160
`-z`	"Suppressing Job Identifier" on page 91

3.13.1 Specifying Queue and/or Server

The "`-q destination`" option to `qsub` allows you to specify a particular destination to which you want the job submitted. The *destination* names a queue, a Server, or a queue at a Server. The `qsub` command will submit the script to the Server defined by the *destination* argument. If the *destination* is a routing queue, the job may be routed by the Server to a new destination. If the `-q` option is not specified, the `qsub` command will submit the script to the default queue at the default Server. (See also the discussion of **`PBS_DEFAULT`** in "Environment Variables" on page 20.) The destination specification takes the following form:

```
-q [queue[@host]]
```

Examples:

```
qsub -q queue my_job
qsub -q @server my_job
#PBS -q queueName
qsub -q queueName@serverName my_job
qsub -q queueName@serverName.domain.com my_job
```

3.13.2 Managing Output and Error Files

3.13.2.1 Default Behavior

PBS, by default, always copies the standard output (stdout) and standard error (stderr) files back to $PBS_O_WORKDIR on the submission host when a job finishes. When qsub is run, it sets $PBS_O_WORKDIR to the current working directory where the qsub command is executed.

3.13.2.2 Specifying Path for Output and Error Files

The "`-o path`" and "`-e path`" options to `qsub` allows you to specify the name of the files to which the stdout and the stderr file streams should be written. The path argument is of the form: `[hostname:]path_name` where *hostname* is the name of a host to which the file will be returned and *path_name* is the path name on that host. You may

specify relative or absolute paths. If you specify only a file name, it is assumed to be relative to your home directory. Do not use variables in the path. The following examples illustrate these various options.

```
#PBS -o /u/user1/myOutputFile
#PBS -e /u/user1/myErrorFile
```

```
qsub -o myOutputFile my_job
qsub -o /u/user1/myOutputFile my_job
qsub -o myWorkstation:/u/user1/myOutputFile my_job
qsub -e myErrorFile my_job
qsub -e /u/user1/myErrorFile my_job
qsub -e myWorkstation:/u/user1/myErrorFile my_job
```

Note that if the PBS client commands are used on a Windows host, then special characters like spaces, backslashes (\), and colons (:) can be used in command line arguments such as for specifying pathnames, as well as drive letter specifications. The following are allowed:

```
qsub -o \temp\my_out job.scr
qsub -e "host:e:\Documents and Settings\user\Desk-
   top\output"
```

The error output of the above job is to be copied onto the `e:` drive on *`host`* using the path "`\Documents and Settings\user\Desk-top\output`". The quote marks are required when arguments to `qsub` contain spaces.

3.13.2.3 Output Appended When Job is Rerun

If your job runs and produces output, and then is rerun, meaning that another job with the same name is run, PBS appends the output of the second run to that of the first. The first output is preserved.

3.13.2.4 Merging Output and Error Files

The "`-j join`" option declares if the standard error stream of the job will be merged with the standard output stream of the job. A *join* argument value of `oe` directs that the two streams will be merged, intermixed, as

standard output. A *join* argument value of eo directs that the two streams will be merged, intermixed, as standard error. If the *join* argument is n or the option is not specified, the two streams will be two separate files.

```
qsub -j oe my_job
#PBS -j eo
```

3.13.2.5 Retaining Output and Error Files on Execution Host

The "-k *keep*" option defines which (if either) of standard output (STDOUT) or standard error (STDERR) of the job will be retained in the job's staging and execution directory on the primary execution host. If set, this option overrides the path name for the corresponding file. If not set, neither file is retained on the execution host. The argument is either the single letter "e" or "o", or the letters "e" and "o" combined in either order. Or the argument is the letter "n". If "-k" is not specified, neither file is retained.

e

The standard error file is to be retained in the job's staging and execution directory on the primary execution host. The job's name will be the default file name given by: *job_name*.e*sequence* where job_name is the name specified for the job, and *sequence* is the sequence number component of the job identifier.

o

The standard output file is to be retained in the job's staging and execution directory on the primary execution host. The file name will be the default file name given by: *job_name*.o*sequence* where job_name is the name specified for the job, and *sequence* is the sequence number component of the job identifier.

eo, oe

Both standard output and standard streams are retained on the primary execution host, in the job's staging and execution directory.

n

Neither file is retained.

qsub -k oe my_job

#PBS -k eo

3.13.3 Exporting Environment Variables

The "`-V`" option declares that all environment variables in the `qsub` command's environment are to be exported to the batch job.

```
qsub -V my_job
#PBS -V
```

3.13.4 Expanding Environment Variables

The "`-v variable_list`" option to `qsub` allows you to specify additional environment variables to be exported to the job. *variable_list* names environment variables from the `qsub` command environment which are made available to the job when it executes. These variables and their values are passed to the job. These variables are added to those already automatically exported. Format: comma-separated list of strings in the form:

-v variable

or

-v variable=value

If a *variable=value* pair contains any commas, the value must be enclosed in single or double quotes, and the *variable=value* pair must be enclosed in the kind of quotes not used to enclose the value. For example:

```
qsub -v DISPLAY,myvariable=32 my_job
qsub -v "var1='A,B,C,D'" job.sh
qsub -v a=10, "var2='A,B'", c=20, HOME=/home/zzz job.sh
```

3.13.5 Specifying Email Notification

The "`-m MailOptions`" defines the set of conditions under which the execution server will send a mail message about the job. The *MailOptions* argument is a string which consists of either the single character "`n`", or one or more of the characters "`a`", "`b`", and "`e`". If no email notification is specified, the default behavior will be the same as for "`-m a`" .

a

Send mail when job is *aborted* by batch system

Example:

```
Job to be deleted at request "root@host."
```

b

Send mail when job *begins* execution

Example:

```
Begun execution
```

e

Send mail when job *ends* execution

Example:

```
Execution terminated
Exit_status=0
resources_used.cpupercent=0
resources_used.cput=00:00:00
resources_used.mem=2464kb
resources_used.ncpus=1
resources_used.vmem=2455kb
resources_used.walltime=00:00:07
```

n

Do not send mail

Examples:

```
qsub -m ae my_job
#PBS -m b
```

3.13.6 Setting Email Recipient List

The "-M user_list" option declares the list of users to whom mail is sent by the execution server when it sends mail about the job. The user_list argument is of the form:

user[@host][,user[@host],...]

If unset, the list defaults to the submitting user at the qsub host, i.e. the job owner.

```
qsub -M user1@mydomain.com my_job
```

3.13.6.1 Caveats

PBS on Windows can only send email to addresses that specify an actual hostname that accepts port 25 (sendmail) requests. For the above example on Windows you will need to specify:

```
qsub -M user1@host.mydomain.com
```

where host.mydomain.com accepts port 25 connections.

3.13.7 Specifying a Job Name

The "-N name" option declares a name for the job. The *name* specified may be any length. The first character must be alphabetic, numeric, hyphen, underscore, or plus sign. If the -N option is not specified, the job name will be the base name of the job script file specified on the command line. If no script file name was specified and the script was read from the standard input, then the job name will be set to STDIN.

```
qsub -N myName my_job
#PBS -N myName
```

3.13.8 Marking a Job as "Rerunnable" or Not

The "`-r y|n`" option declares whether the job is rerunnable. To rerun a job is to terminate the job and requeue it in the execution queue in which the job currently resides. The *value* argument is a single character, either "`y`" or "`n`". If the argument is "`y`", the job is rerunnable. If the argument is "`n`", the job is not rerunnable. The default value is "`y`", rerunnable.

```
qsub -r n my_job
#PBS -r n
```

Marking your job as non-rerunnable will not affect how PBS treats it in the case of startup failure. If a job that is marked non-rerunnable has an error during startup, before it begins execution, that job is requeued for another attempt. The purpose of marking a job as non-rerunnable is to prevent it from running twice and using data that undergoes a change during execution. However, if the job never actually starts execution, the data isn't altered before the job uses it, so PBS requeues it.

PBS requeues some jobs that are terminated before execution. Two examples of this are multi-host jobs where the job did not start on one or more execution hosts, and provisioning jobs for which the provisioning script failed.

3.13.9 Specifying Scripting Language to Use

The "`-S path_list`" option declares the path and name of the scripting language to be used in interpreting the job script. The option argument *path_list* is in the form: `path[@host][,path[@host],...]` Only one path may be specified for any host named, and only one path may be specified without the corresponding host name. The path selected will be the one with the host name that matched the name of the execution host. If no matching host is found, then the path specified without a host will be selected, if present. If the `-S` option is not specified, the option argument is the null string, or no entry from the *path_list* is selected, then PBS will use the user's login shell on the execution host.

Example 1: Using `bash` via a directive:

```
#PBS -S /bin/bash@mars,/usr/bin/bash@jupiter
```

Example 2: Running a Python script from the command line on UNIX/Linux:

```
qsub -S /opt/pbs/default/bin/pbs_python <script name>
```

Example 3: Running a Python script from the command line on Windows:

```
qsub -S "C:\Program Files\PBS
   Pro\exec\bin\pbs_python.exe" <script name>
```

3.13.9.1 Windows Caveats

Using this option under Windows is more complicated because if you change from the default shell of `cmd`, then a valid `PATH` is not automatically set. Thus if you use the "`-S`" option under Windows, you must explicitly set a valid `PATH` as the first line of your job script.

3.13.10 Setting a Job's Priority

The "`-p priority`" option defines the priority of the job. The *priority* argument must be an integer between -1024 (lowest priority) and +1023 (highest priority) inclusive. The default is no priority which is equivalent to a priority of zero.

This option allows the user to specify a priority for their jobs. However, this option is dependant upon the local scheduling policy. By default the "sort jobs by job-priority" feature is disabled. If your local PBS administrator has enabled it, then all queued jobs will be sorted based on the user-specified priority. (If you need an absolute ordering of your own jobs, see "Specifying Job Dependencies" on page 161.)

```
qsub -p 120 my_job
```

```
#PBS -p -300
```

3.13.11 Deferring Execution

The "`-a date_time`" option declares the time after which the job is eligible for execution. The *date_time* argument is in the form:

[[[[CC]YY]MM]DD]hhmm[.SS]

where

CC is the first two digits of the year (the century)

YY is the second two digits of the year

MM is the two digits for the month

DD is the day of the month

hh is the hour

mm is the minute

The optional *SS* is the seconds

If the month, *MM*, is not specified, it will default to the current month if the specified day *DD*, is in the future. Otherwise, the month will be set to next month. Likewise, if the day, DD, is not specified, it will default to today if the time *hhmm* is in the future. Otherwise, the day will be set to tomorrow. For example, if you submit a job at 11:15am with a time of "1110", the job will be eligible to run at 11:10am tomorrow. Other examples include:

```
qsub -a 0700 my_job
#PBS -a 10220700
```

The job is in the wait (W) state from the time it is submitted until the time it is eligible for execution.

3.13.12 Holding a Job (Delaying Execution)

The "-h" option specifies that a *user hold* be applied to the job at submission time. The job will be submitted, then placed in a hold state. The job will remain ineligible to run until the hold is released. (For details on releasing a held job see "Holding and Releasing Jobs" on page 119.)

```
qsub -h my_job
#PBS -h
```

3.13.13 Specifying Job Checkpoint Interval

3.13.13.1 Checkpointable Jobs

A job is checkpointable if any of the following is true:

- Its application supports checkpointing and there are checkpoint scripts
- There is a third-party checkpointing application available
- The OS supports checkpointing

Checkpoint scripts are set up by the local system administrator.

3.13.13.2 Queue Checkpoint Intervals

The execution queue in which the job resides controls the minimum interval at which a job can be checkpointed. The interval is specified in CPU minutes or walltime minutes. The same value is used for both, so for example if the minimum interval is specified as 12, then a job using the queue's interval for CPU time will be checkpointed every 12 minutes of CPU time, and a job using the queue's interval for walltime will be checkpointed every 12 minutes of walltime.

3.13.13.3 Checkpoint Interval

The "`-c checkpoint-spec`" option defines the interval, in CPU minutes, or in walltime minutes, at which the job will be checkpointed.

The *checkpoint-spec* argument is specified as:

c
: Checkpointing is to be performed according to the interval, measured in CPU time, set on the execution queue in which the job resides.

c=<minutes of CPU time>
: Checkpointing is to be performed at intervals of the specified number of minutes of CPU time used by the job. This value must be greater than zero. If the interval specified is less than that set on the execution queue in which the job resides, the queue's interval is used.

Format: Integer

w

Checkpointing is to be performed according to the interval, measured in walltime, set on the execution queue in which the job resides.

w=<minutes of walltime>

Checkpointing is to be performed at intervals of the specified number of minutes of walltime used by the job. This value must be greater than zero. If the interval specified is less than that set on the execution queue in which the job resides, the queue's interval is used.

Format: Integer

n

No checkpointing is to be performed.

s

Checkpointing is to be performed only when the Server executing the job is shut down.

u

Checkpointing is unspecified, thus resulting in the same behavior as "s".

If "`-c`" is not specified, the checkpoint attribute is set to the value "`u`".

```
qsub -c c my_job
#PBS -c c=10
```

Checkpointing is not supported for job arrays.

3.13.14 Specifying Job User ID

PBS requires that a user's name be consistent across a server and its execution hosts, but not across a submission host and a server. A user may have access to more than one server, and may have a different username on each server. In this environment, if a user wishes to submit a job to any of the available servers, the username for each server is specified. The wildcard username will be used if the job ends up at yet another server not specified, but only if that wildcard username is valid.

For example, our user is UserS on the submission host HostS, UserA on server ServerA, and UserB on server ServerB, and is UserC everywhere else. Note that this user must be UserA on all ExecutionA and UserB on all ExecutionB machines. Then our user can use "qsub -u UserA@ServerA,UserB@ServerB,UserC" for the job. The job owner will always be UserS.

Usernames are limited to 15 characters.

3.13.14.1 `qsub -u`: User ID with UNIX

The server's flatuid attribute determines whether it assumes that identical usernames mean identical users. If true, it assumes that if UserS exists on both the submission host and the server host, then UserS can run jobs on that server. If not true, the server calls ruserok() which uses /etc/hosts.equiv and .rhosts to authorize UserS to run as UserS.

Table 3-4: UNIX User ID and flatuid

Value of flatuid	Submission host username/server host username	
	Same: UserS/UserS	Different: UserS/UserA
True	Server assumes user has permission to run job	Server checks whether UserS can run job as UserA
Not true	Server checks whether UserS can run job as UserS	Server checks whether UserS can run job as UserA

Note that if different names are listed via the `-u` option, then they are checked regardless of the value of `flatuid`.

3.13.14.2 `qsub -u`: User ID with Windows

Under Windows, if a user has a non-admin account, the server's hosts.equiv file is used to determine whether that user can run a job on a given server. For an admin account, [PROFILE_PATH].\rhosts is used, and the server's acl_roots attribute must be set to allow job submissions.

Usernames containing spaces are allowed as long as the username length is no more than 15 characters, and the usernames are quoted when used in the command line.

Table 3-5: Requirements for Admin User to Submit Job

Location/Action	Submission host username/Server host username	
	Same: UserS/UserS	Different: UserS/ UserA
[PROFILE_PATH]\ .rhosts contains	For UserS on ServerA, add <HostS> UserS	For UserA on ServerA, add <HostS> UserS
set ServerA's acl_roots attribute	qmgr> set server acl_roots=UserS	qmgr> set server acl_roots=UserA

Table 3-6: Requirements for Non-admin User to Submit Job

File	Submission host username/Server host username	
	Same: UserS/UserS	Different: UserS/UserA
hosts.equiv on ServerA	<HostS>	<HostS> UserS

3.13.15 Setting Job Attribute Values

You can use the `-W <attribute>=<value>` option to the `qsub` command to set any job attribute. This option duplicates the function of several other qsub options. For example, using "`-e <path>`" or "`-W Error_Path=<path>`" has the same effect.

Avoid duplicating other `qsub` options when using the `-W` option.

3.13.16 Specifying Job Group ID

The "`-W group_list=g_list`" option defines the group name under which the job is to run on the execution system. The *g_list* argument is of the form:

```
group[@host][,group[@host],...]
```

Only one group name may be given per specified host. Only one of the group specifications may be supplied without the corresponding host specification. That group name will used for execution on any host not named in the argument list. If not set, the *group_list* defaults to the primary group of the user under which the job will be run. Under Windows, the primary group is the first group found for the user by PBS when querying the accounts database.

qsub -W *group_list=grpA,grpB@jupiter* my_job

3.13.17 Specifying a Local Account

The "`-A account_string`" option defines the account string associated with the job. The *account_string* is an opaque string of characters and is not interpreted by the Server which executes the job. This value is often used by sites to track usage by locally defined account names.

IMPORTANT:

Under Unicos, if the Account string is specified, it must be a valid account as defined in the system "User Data Base", UDB.

qsub -A *Math312* my_job

```
#PBS -A accountNumber
```

3.13.18 Suppressing Job Identifier

The "-z" option directs the qsub command to not write the job identifier assigned to the job to the command's standard output.

```
qsub -z my_job
#PBS -z
```

3.13.19 Specifying Staging and Execution Directory

The -W sandbox=<value> option allows you to specify where PBS will stage files and execute the job script. See section 7.6 "Input/Output File Staging" on page 166.

3.13.20 Interactive-batch Jobs

PBS provides a special kind of batch job called *interactive-batch*. An interactive-batch job is treated just like a regular batch job (in that it is queued up, and has to wait for resources to become available before it can run). Once it is started, however, the user's terminal input and output are connected to the job in a matter similar to a login session. It appears that the user is logged into one of the available execution machines, and the resources requested by the job are reserved for that job. Many users find this useful for debugging their applications or for computational steering. The "qsub -I" option declares that the job is an interactive-batch job.

Interactive jobs can use provisioning.

If the qsub -I option is specified on the command line, the job is an interactive job. If a script is given, it will be processed for directives, but any executable commands will be discarded. When the job begins execution, all input to the job is from the terminal session in which qsub is running. The -I option is ignored in a script directive.

When an interactive job is submitted, the qsub command will not terminate when the job is submitted. qsub will remain running until the job terminates, is aborted, or the user interrupts qsub with a SIGINT (the

control-C key). If `qsub` is interrupted prior to job start, it will query if the user wishes to exit. If the user responds “yes”, `qsub` exits and the job is aborted.

Once the interactive job has started execution, input to and output from the job pass through `qsub`. Keyboard-generated interrupts are passed to the job. Lines entered that begin with the tilde ('~') character and contain special sequences are interpreted by `qsub` itself. The recognized special sequences are:

~.
: `qsub` terminates execution. The batch job is also terminated.

~susp
: If running under the UNIX C shell, suspends the `qsub` program. “susp” is the suspend character, usually CNTL-Z.

~asusp
: If running under the UNIX C shell, suspends the input half of `qsub` (terminal to job), but allows output to continue to be displayed. “asusp” is the auxiliary suspend character, usually control-Y.

3.13.20.1 Caveats

- Interactive-batch jobs are not supported on Windows.
- Interactive-batch jobs do not support job arrays.

3.14 Failed Jobs

Once a job has experienced a certain number of failures, PBS holds the job.

If requeueing a job fails, the job is deleted.

Chapter 4

Using the xpbs GUI

The PBS graphical user interface is called **xpbs**, and provides a user-friendly, point and click interface to the PBS commands. xpbs utilizes the tcl/tk graphics tool suite, while providing the user with most of the same functionality as the PBS CLI commands. In this chapter we introduce xpbs, and show how to create a PBS job using xpbs.

4.1 Using the xpbs command

4.1.1 Starting xpbs

If PBS is installed on your local workstation, or if you are running under Windows, you can launch xpbs by double-clicking on the xpbs icon on the desktop. You can also start xpbs from the command line with the following command.

UNIX:

```
xpbs &
```

Windows:

```
xpbs.exe
```

Doing so will bring up the main xpbs window, as shown below.

4.1.2 Running xpbs Under UNIX

Before running xpbs for the first time under UNIX, you may need to configure your workstation for it. Depending on how PBS is installed at your site, you may need to allow xpbs to be displayed on your workstation. However, if the PBS client commands are installed locally on your workstation, you can skip this step. (Ask your PBS administrator if you are unsure.)

The most secure method of running xpbs remotely and displaying it on your local XWindows session is to redirect the XWindows traffic through ssh (secure shell), via setting the "X11Forwarding yes" parameter in the sshd_config file. (Your local system administrator can provide details on this process if needed.)

An alternative, but less secure, method is to direct your X-Windows session to permit the xpbs client to connect to your local X-server. Do this by running the xhost command with the name of the host from which you will be running xpbs, as shown in the example below:

```
xhost + server.mydomain.com
```

Next, on the system from which you will be running xpbs, set your X-Windows **DISPLAY** variable to your local workstation. For example, if using the C-shell:

```
setenv DISPLAY myWorkstation:0.0
```

However, if you are using the Bourne or Korn shell, type the following:

```
export DISPLAY=myWorkstation:0.0
```

4.2 Using xpbs: Definitions of Terms

The various panels, boxes, and regions (collectively called “widgets”) of `xpbs` and how they are manipulated are described in the following sections. A *listbox* can be multi-selectable (a number of entries can be selected/highlighted using a mouse click) or single-selectable (one entry can be highlighted at a time).

For a multi-selectable listbox, the following operations are allowed:

- left-click to select/highlight an entry.
- shift-left-click to contiguously select more than one entry.
- control-left-click to select multiple non-contiguous entries.
- click the *Select All / Deselect All* button to select all entries or deselect all entries at once.
- double clicking an entry usually activates some action that uses the selected entry as a parameter.

An *entry* widget is brought into focus with a left-click. To manipulate this widget, simply type in the text value. Use of arrow keys and mouse selection of text for deletion, overwrite, copying and pasting with sole use of mouse buttons are permitted. This widget has a scrollbar for horizontally scanning a long text entry string.

A *matrix of entry boxes* is usually shown as several rows of entry widgets where a number of entries (called fields) can be found per row. The matrix is accompanied by up/down arrow buttons for paging through the rows of data, and each group of fields gets one scrollbar for horizontally scanning long entry strings. Moving from field to field can be done using the <Tab> (move forward), <Cntrl-f> (move forward), or <Cntrl-b> (move backward) keys.

A *spinbox* is a combination of an entry widget and a horizontal scrollbar. The entry widget will only accept values that fall within a defined list of valid values, and incrementing through the valid values is done by clicking on the up/down arrows.

A *button* is a rectangular region appearing either raised or pressed that invokes an action when clicked with the left mouse button. When the button appears pressed, then hitting the <RETURN> key will automatically select the button.

A *text region* is an editor-like widget. This widget is brought into focus with a left-click. To manipulate this widget, simply type in the text. Use of arrow keys, backspace/delete key, mouse selection of text for deletion or overwrite, and copying and pasting with sole use of mouse buttons are permitted. This widget has a scrollbar for vertically scanning a long entry.

4.3 Introducing the xpbs Main Display

The main window or display of `xpbs` is comprised of five collapsible subwindows or *panels*. Each panel contains specific information. Top to bottom, these panels are: the Menu Bar, Hosts panel, Queues panel, Jobs panel, and the Info panel.

4.3.1 xpbs Menu Bar

The Menu Bar is composed of a row of command buttons that signal some action with a click of the left mouse button. The buttons are:

Manual Update
: forces an update of the information on hosts, queues, and jobs.

Auto Update
: sets an automatic update of information every user-specified number of minutes.

Track Job
: for periodically checking for returned output files of jobs.

Preferences

for setting parameters such as the list of Server host(s) to query.

Help

contains some help information.

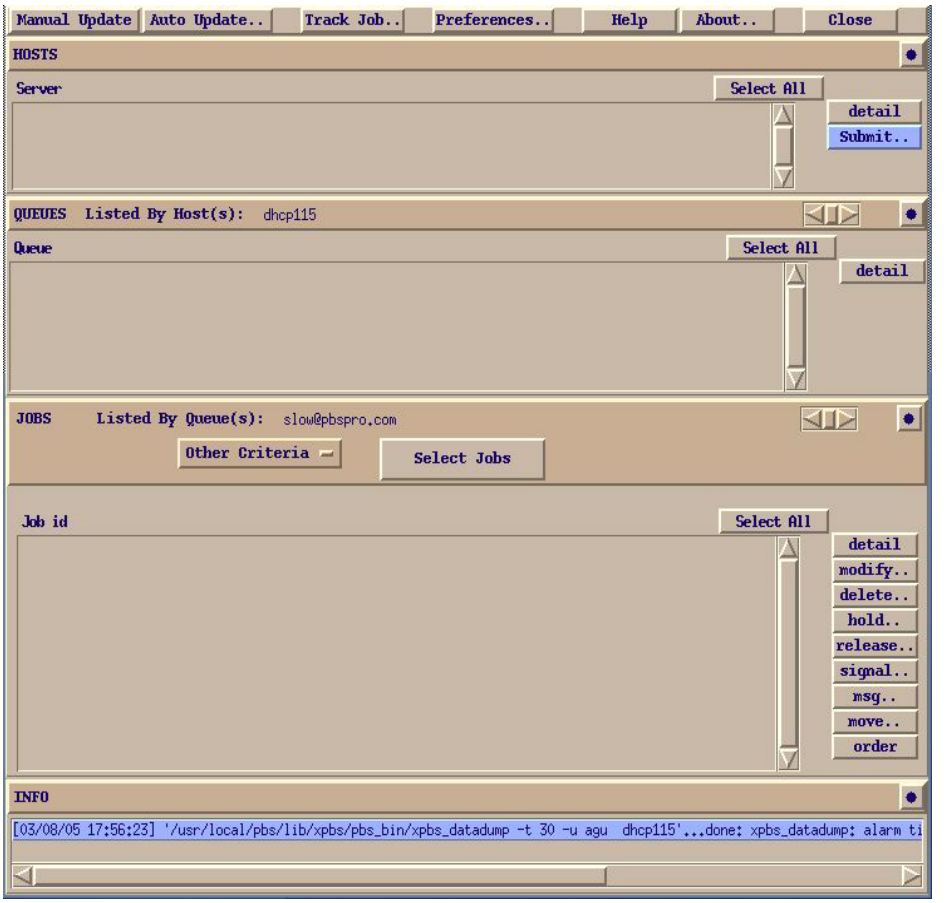
About

gives general information about the xpbs GUI.

Close

for exiting `xpbs` plus saving the current setup information.

.



4.3.2 xpbs Hosts Panel

The Hosts panel is composed of a leading horizontal HOSTS bar, a listbox, and a set of command buttons. The HOSTS bar contains a minimize/maximize button, identified by a dot or a rectangular image, for displaying or iconizing the Hosts region. The listbox displays information about favorite Server host(s), and each entry is meant to be selected via a single left-click, shift-left-click for contiguous selection, or control-left-click for non-contiguous selection.

To the right of the Hosts Panel are buttons that represent actions that can be performed on selected host(s). Use of these buttons will be explained in detail below.

detail

Provides information about selected Server host(s). This functionality can also be achieved by double clicking on an entry in the Hosts listbox.

submit

For submitting a job to any of the queues managed by the selected host(s).

terminate

For terminating (shutting down) PBS Servers on selected host(s). (Visible via the "`-admin`" option only.)

IMPORTANT:

Note that some buttons are only visible if `xpbs` is started with the "`-admin`" option, which requires manager or operator privilege to function.

The middle portion of the Hosts Panel has abbreviated column names indicating the information being displayed, as the following table shows:

Table 4-1: xpbs Server Column Headings

Heading	Meaning
Max	Maximum number of jobs permitted
Tot	Count of jobs currently enqueued in any state
Que	Count of jobs in the Queued state

Table 4-1: xpbs Server Column Headings

Heading	Meaning
Run	Count of jobs in the Running state
Hld	Count of jobs in the Held state
Wat	Count of jobs in the Waiting state
Trn	Count of jobs in the Transiting state
Ext	Count of jobs in the Exiting state
Status	Status of the corresponding Server
PEsInUse	Count of Processing Elements (CPUs, PEs, Vnodes) in Use

4.3.3 xpbs Queues Panel

The Queues panel is composed of a leading horizontal QUEUES bar, a listbox, and a set of command buttons. The QUEUES bar lists the hosts that are consulted when listing queues; the bar also contains a minimize/maximize button for displaying or iconizing the Queues panel. The listbox displays information about queues managed by the Server host(s) selected from the Hosts panel; each listbox entry can be selected as described above for the Hosts panel.

To the right of the Queues Panel area are buttons for actions that can be performed on selected queue(s).

detail
: provides information about selected queue(s). This functionality can also be achieved by double clicking on a Queue listbox entry.

stop
: for stopping the selected queue(s). (`-admin` only)

start
: for starting the selected queue(s). (`-admin` only)

disable
: for disabling the selected queue(s). (`-admin` only)

enable
for enabling the selected queue(s). (`-admin` only)

The middle portion of the Queues Panel has abbreviated column names indicating the information being displayed, as the following table shows:

Table 4-2: xpbs Queue Column Headings

Heading	Meaning
Max	Maximum number of jobs permitted
Tot	Count of jobs currently enqueued in any state
Ena	Is queue enabled? yes or no
Str	Is queue started? yes or no
Que	Count of jobs in the Queued state
Run	Count of jobs in the Running state
Hld	Count of jobs in the Held state
Wat	Count of jobs in the Waiting state
Trn	Count of jobs in the Transiting state
Ext	Count of jobs in the Exiting state
Type	Type of queue: execution or route
Server	Name of Server on which queue exists

4.3.4 xpbs Jobs Panel

The Jobs panel is composed of a leading horizontal JOBS bar, a listbox, and a set of command buttons. The JOBS bar lists the queues that are consulted when listing jobs; the bar also contains a minimize/maximize button for displaying or iconizing the Jobs region. The listbox displays information about jobs that are found in the queue(s) selected from the Queues listbox; each listbox entry can be selected as described above for the Hosts panel.

The region just above the Jobs listbox shows a collection of command buttons whose labels describe criteria used for filtering the Jobs listbox contents. The list of jobs can be selected according to the owner of jobs (Owners), job state (Job_States), name of the job (Job_Name), type of hold placed on the job (Hold_Types), the account name associated with the job (Account_Name), checkpoint attribute (Checkpoint), time the job is eligible for queueing/execution (Queue_Time), resources requested by the job (Resources), priority attached to the job (Priority), and whether or not the job is rerunnable (Rerunnable).

The selection criteria can be modified by clicking on any of the appropriate command buttons to bring up a selection box. The criteria command buttons are accompanied by a *Select Jobs* button, which when clicked, will update the contents of the Jobs listbox based on the new selection criteria. Note that only jobs that meet ***all*** the selected criteria will be displayed.

Finally, to the right of the Jobs panel are the following command buttons, for operating on selected job(s):

detail
: provides information about selected job(s). This functionality can also be achieved by double-clicking on a Jobs listbox entry.

modify
: for modifying attributes of the selected job(s).

delete
: for deleting the selected job(s).

hold
: for placing some type of hold on selected job(s).

release
: for releasing held job(s).

signal
: for sending signals to selected job(s) that are running.

msg
: for writing a message into the output streams of selected job(s).

move
: for moving selected job(s) into some specified destination.

order
for exchanging order of two selected jobs in a queue.

run
for running selected job(s). (-admin only)

rerun
for requeueing selected job(s) that are running. (-admin only)

The middle portion of the Jobs Panel has abbreviated column names indicating the information being displayed, as the following table shows:

Table 4-3: xpbs Job Column Headings

Heading	Meaning
Job id	Job Identifier
Name	Name assigned to job, or script name
User	User name under which job is running
PEs	Number of Processing Elements (CPUs) requested
CputUse	Amount of CPU time used
WalltUse	Amount of wall-clock time used
S	State of job
Queue	Queue in which job resides

4.3.5 xpbs Info Panel

The Info panel shows the progress of the commands executed by `xpbs`. Any errors are written to this area. The INFO panel also contains a minimize/maximize button for displaying or iconizing the Info panel.

4.3.6 xpbs Keyboard Tips

There are a number of shortcuts and key sequences that can be used to speed up using `xpbs`. These include:

Tip 1.

All buttons which appear to be depressed in the dialog box/ subwindow can be activated by pressing the return/enter key.

Tip 2.

Pressing the tab key will move the blinking cursor from one text field to another.

Tip 3.

To contiguously select more than one entry: left-click then drag the mouse across multiple entries.

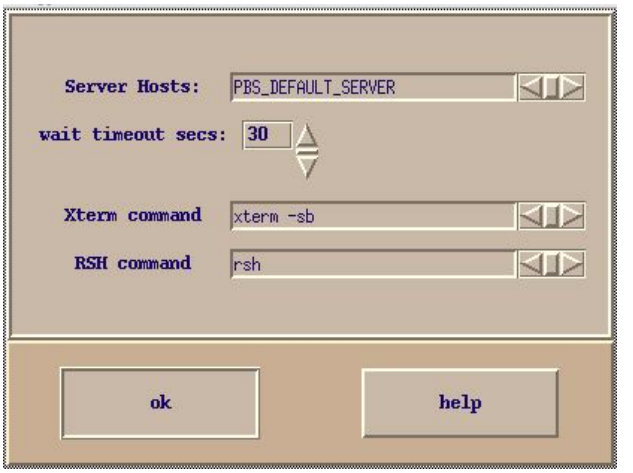
Tip 4.

To non-contiguously select more than one entry: hold the control-left-click on the desired entries.

4.4 Setting xpbs Preferences

The “Preferences” button is in the Menu Bar at the top of the main `xpbs` window. Clicking it will bring up a dialog box that allows you to customize the behavior of xpbs:

1. Define Server hosts to query
2. Select wait timeout in seconds
3. Specify `xterm` command (for interactive jobs, UNIX only)
4. Specify which `rsh`/`ssh` command to use



4.5 Relationship Between PBS and xpbs

xpbs is built on top of the PBS client commands, such that all the features of the command line interface are available through the GUI. Each "task" that you perform using xpbs is converted into the necessary PBS command and then run.

Table 4-4: xpbs Buttons and PBS Commands

Location	Command Button	PBS Command
Hosts Panel	detail	`qstat -B -f` *selected server_host(s)*
Hosts Panel	submit	`qsub` *options selected Server(s)*
Hosts Panel	terminate *	`qterm` *selected server_host(s)*
Queues Panel	detail	`qstat -Q -f` *selected queue(s)*
Queues Panel	stop *	`qstop` *selected queue(s)*
Queues Panel	start *	`qstart` *selected queue(s)*
Queues Panel	enable *	`qenable` *selected queue(s)*
Queues Panel	disable *	`qdisable` *selected queue(s)*
Jobs Panel	detail	`qstat -f` *selected job(s)*
Jobs Panel	modify	`qalter` *selected job(s)*
Jobs Panel	delete	`qdel` *selected job(s)*
Jobs Panel	hold	`qhold` *selected job(s)*
Jobs Panel	release	`qrls` *selected job(s)*
Jobs Panel	run	`qrun` *selected job(s)*
Jobs Panel	rerun	`qrerun` *selected job(s)*
Jobs Panel	signal	`qsig` *selected job(s)*

Table 4-4: `xpbs` Buttons and PBS Commands

Location	Command Button	PBS Command
Jobs Panel	msg	`qmsg` *selected job(s)*
Jobs Panel	move	`qmove` *selected job(s)*
Jobs Panel	order	`qorder` *selected job(s)*

* Indicates command button is visible only if `xpbs` is started with the "`-admin`" option.

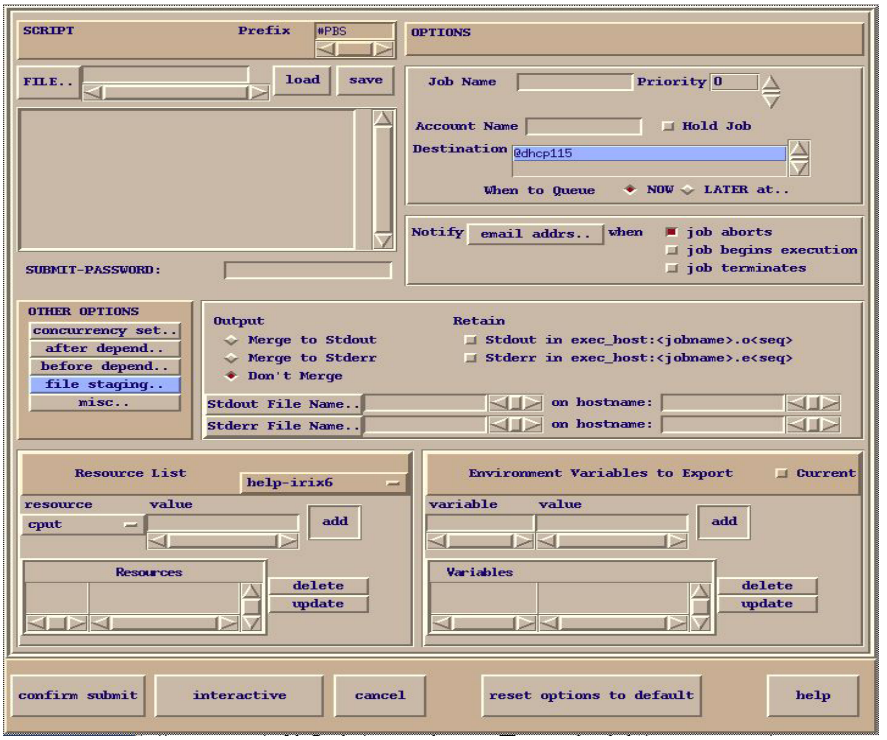
4.6 How to Submit a Job Using `xpbs`

To submit a job using `xpbs`, perform the following steps:

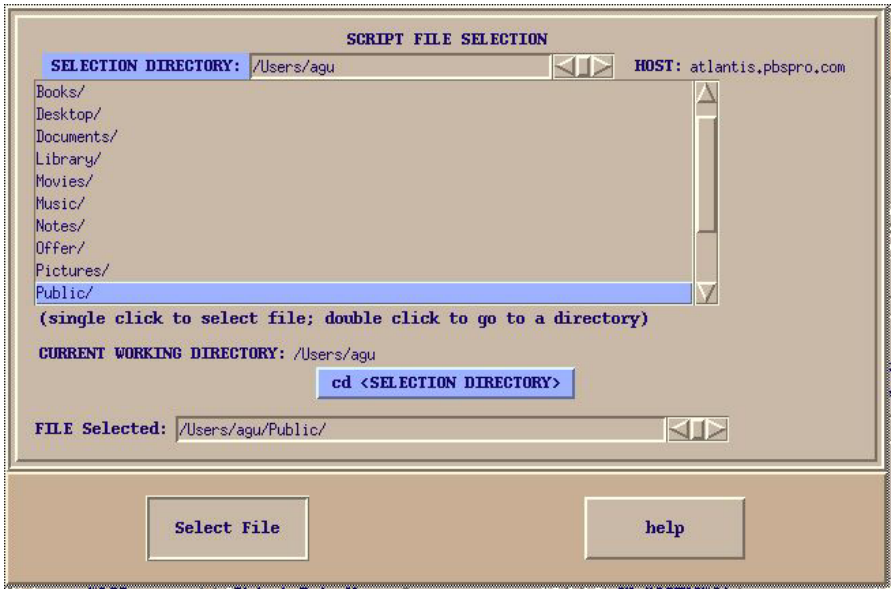
First, select a host from the HOSTS listbox in the main `xpbs` display to which you wish to submit the job.

Next, click on the *Submit* button located next to the HOSTS panel. The *Submit* button brings up the Submit Job Dialog box (see below) which is composed of four distinct regions. The Job Script File region is at the upper left. The OPTIONS region containing various widgets for setting job

attributes is scattered all over the dialog box. The OTHER OPTIONS is located just below the Job Script file region, and COMMAND BUTTONS region is at the bottom.



The job script region is composed of a header box, the text box, FILE entry box, and two buttons labeled *load* and *save.* If you have a script file containing PBS options and executable lines, then type the name of the file on the FILE entry box, and then click on the *load* button. Alternatively, you may click on the *FILE* button, which will display a File Selection browse window, from which you may point and click to select the file you wish to open. The File Selection Dialog window is shown below. Clicking on the *Select File* button will load the file into `xpbs`, just as does the *load* button described above.



The various fields in the Submit window will get loaded with values found in the script file. The script file text box will only be loaded with executable lines (non-PBS) found in the script. The job script header box has a *Prefix* entry box that can be modified to specify the PBS directive to look for when parsing a script file for PBS options.

If you don't have a existing script file to load into `xpbs`, you can start typing the executable lines of the job in the file text box.

Next, review the Destination listbox. This box shows the queues found in the host that you selected. A special entry called "`@host`" refers to the default queue at the indicated host. Select appropriately the destination queue for the job.

Next, define any required resources in the Resource List subwindow.

The resources specified in the "Resource List" section will be job-wide resources only. In order to specify chunks or job placement, use a script.

To run an array job, use a script. You will not be able to query individual subjobs or the whole job array using xpbs. Type the script into the "File: entry" box. Do not click the "Load" button. Instead, use the "Submit" button.

Finally, review the optional settings to see if any should apply to this job.

For example:

- Use the one of the buttons in the "Output" region to merge output and error files.
- Use "Stdout File Name" to define standard output file and to redirect output
- Use the "Environment Variables to Export" subwindow to have current environment variables exported to the job.
- Use the "Job Name" field in the OPTIONS subwindow to give the job a name.
- Use the "Notify email address" and one of the buttons in the OPTIONS subwindow to have PBS send you mail when the job terminates.

Now that the script is built you have four options of what to do next:

Reset options to default

Save the script to a file

Submit the job as a batch job

Submit the job as an interactive-batch job (UNIX only)

Reset clears all the information from the submit job dialog box, allowing you to create a job from a fresh start.

Use the FILE. field (in the upper left corner) to define a filename for the script. Then press the *Save* button. This will cause a PBS script file to be generated and written to the named file.

Pressing the *Confirm Submit* button at the bottom of the Submit window will submit the PBS job to the selected destination. `xpbs` will display a small window containing the job identifier returned for this job. Clicking *OK* on this window will cause it and the Submit window to be removed from your screen.

On UNIX systems (not Windows) you can alternatively submit the job as an interactive-batch job, by clicking the *Interactive* button at the bottom of the Submit Job window. Doing so will cause an X-terminal window (`xterm`) to be launched, and within that window a PBS interactive-batch job submitted. The path for the xterm command can be set via the prefer-

ences, as discussed above in section 4.4 “Setting xpbs Preferences” on page 104. For further details on usage, and restrictions, see section 3.13.20 “Interactive-batch Jobs” on page 91.)

4.7 Exiting xpbs

Click on the *Close* button located in the Menu bar to leave `xpbs`. If any settings have been changed, `xpbs` will bring up a dialog box asking for a confirmation in regards to saving state information. The settings will be saved in the `.xpbsrc` configuration file, and will be used the next time you run `xpbs`, as discussed in the following section.

4.8 The xpbs Configuration File

Upon exit, the `xpbs` state may be written to the `.xpbsrc` file in the user’s home directory. (See also section 2.14.2 “Windows User's HOMEDIR” on page 17.) Information saved includes: the selected host(s), queue(s), and job(s); the different jobs listing criteria; the view states (i.e. minimized/maximized) of the Hosts, Queues, Jobs, and INFO regions; and all settings in the Preferences section. In addition, there is a system-wide `xpbs` configuration file, maintained by the PBS Administrator, which is used in the absence of a user’s personal `.xpbsrc` file.

4.9 `xpbs` Preferences

The resources that can be set in the `xpbs` configuration file, `~/.xpbsrc`, are:

***serverHosts**

List of Server hosts (space separated) to query by `xpbs`. A special keyword **`PBS_DEFAULT_SERVER`** can be used which will be used as a placeholder for the value obtained

from the `/etc/pbs.conf` file (UNIX) or "`[PBS Destination Folder]\pbs.conf`" file (Windows).

***timeoutSecs**
Specify the number of seconds before timing out waiting for a connection to a PBS host.

***xtermCmd**
The xterm command to run driving an interactive PBS session.

***labelFont**
Font applied to text appearing in labels.

***fixlabelFont**
Font applied to text that label fixed-width widgets such as listbox labels. This must be a fixed-width font.

***textFont**
Font applied to a text widget. Keep this as fixed-width font.

***backgroundColor**
The color applied to background of frames, buttons, entries, scrollbar handles.

***foregroundColor**
The color applied to text in any context.

***activeColor**
The color applied to the background of a selection, a selected command button, or a selected scroll bar handle.

***disabledColor**
Color applied to a disabled widget.

***signalColor**
Color applied to buttons that signal something to the user about a change of state. For example, the

color of the *Track Job* button when returned output files are detected.

***shadingColor**
A color shading applied to some of the frames to emphasize focus as well as decoration.

***selectorColor**
The color applied to the selector box of a radiobutton or checkbutton.

***selectHosts**
List of hosts (space separated) to automatically select/highlight in the HOSTS listbox.

***selectQueues**
List of queues (space separated) to automatically select/highlight in the QUEUES listbox.

***selectJobs**
List of jobs (space separated) to automatically select/highlight in the JOBS listbox.

***selectOwners**
List of owners checked when limiting the jobs appearing on the Jobs listbox in the main `xpbs` window. Specify value as "Owners: <list_of_owners>". See `-u` option in `qselect(1B)` for format of <list_of_owners>.

***selectStates**
List of job states to look for (do not space separate) when limiting the jobs appearing on the Jobs listbox in the main `xpbs` window. Specify value as "Job_States: <states_string>". See `-s` option in `qselect(1B)` for format of <states_string>.

***selectRes**
List of resource amounts (space separated) to consult when limiting the jobs appearing on the Jobs

listbox in the main `xpbs` window. Specify value as "Resources: <res_string>". See `-l` option in `qselect(1B)` for format of <res_string>.

***selectExecTime**
The Execution Time attribute to consult when limiting the list of jobs appearing on the Jobs listbox in the main `xpbs` window. Specify value as "Queue_Time: <exec_time>". See `-a` option in `qselect(1B)` for format of <exec_time>.

***selectAcctName**

The name of the account that will be checked when limiting the jobs appearing on the Jobs listbox in the main `xpbs` window. Specify value as "Account_Name: <account_name>". See `-A` option in `qselect(1B)` for format of <account_name>.

***selectCheckpoint**

The checkpoint attribute relationship (including the logical operator) to consult when limiting the list of jobs appearing on the Jobs listbox in the main `xpbs` window. Specify value as "Checkpoint: <checkpoint_arg>". See `-c` option in `qselect(1B)` for format of <checkpoint_arg>.

***selectHold**

The hold types string to look for in a job when limiting the jobs appearing on the Jobs listbox in the main `xpbs` window. Specify value as "Hold_Types: <hold_string>". See `-h` option in `qselect(1B)` for format of <hold_string>.

***selectPriority**

The priority relationship (including the logical operator) to consult when limiting the list of jobs appearing on the Jobs listbox in the main `xpbs` window. Specify value as "Priority: <priority_value>". See `-p` option in `qselect(1B)` for format of <priority_value>.

***selectRerun**

The rerunnable attribute to consult when limiting the list of jobs appearing on the Jobs listbox in the main `xpbs` window. Specify value as "Rerunnable: <rerun_val>". See `-r` option in `qselect(1B)` for format of <rerun_val>.

***selectJobName**

Name of the job that will be checked when limiting the jobs appearing on the Jobs listbox in the main `xpbs` window. Specify value as "Job_Name: <jobname>". See `-N` option in `qselect(1B)` for format of <jobname>.

***iconizeHostsView**

A boolean value (true or false) indicating whether or not to iconize the HOSTS region.

***iconizeQueuesView**
A boolean value (true or false) indicating whether or not to iconize the QUEUES region.

***iconizeJobsView**
A boolean value (true or false) indicating whether or not to iconize the JOBS region.

***iconizeInfoView**
A boolean value (true or false) indicating whether or not to iconize the INFO region.

***jobResourceList**
A curly-braced list of resource names as according to architecture known to `xpbs`. The format is as follows:

{ <arch-type1> resname1 resname2 ... resnameN }

{ <arch-type2> resname1 resname2 ... resnameN }

{ <arch-typeN> resname1 resname2 ... resnameN }

Chapter 5

Working With PBS Jobs

This chapter introduces the reader to various commands useful in working with PBS jobs. Covered topics include: modifying job attributes, holding and releasing jobs, sending messages to jobs, changing order of jobs within a queue, sending signals to jobs, and deleting jobs. In each section below, the command line method for accomplishing a particular task is presented first, followed by the `xpbs` method.

5.1 Modifying Job Attributes

Most attributes can be changed by the owner of the job (or a manager or operator) while the job is still queued. However, once a job begins execution, the only resources that can be modified are `cputime` and `walltime`. These can only be reduced.

When the qalter "-l" option is used to alter the resource list of a queued job, it is important to understand the interactions between altering the select directive and job limits.

If the job was submitted with an explicit "-l select=", then vnode-level resources must be qaltered using the "-l select=" form. In this case a vnode level resource RES cannot be qaltered with the "-l RES" form.

For example:

Submit the job:

```
% qsub -l select=1:ncpus=2:mem=512mb jobscript
```

Job's ID is 230

qalter the job using "-l RES" form:

```
% qalter -l ncpus=4 230
```

Error reported by qalter:

```
qalter: Resource must only appear in "select"
```

specification when select is used: ncpus 230

qalter the job using the "-l select=" form:

```
% qalter -l select=1:ncpus=4:mem=512mb 230
```

No error reported by qalter:

```
%
```

5.1.1 Changing the Selection Directive

If the selection directive is altered, the job limits for any consumable resource in the directive are also modified.

For example, if a job is queued with the following resource list:

`select=2:ncpus=1:mem=5gb, ncpus=2, mem=10gb`

and the selection directive is altered to request

`select=3:ncpus=2:mem=6gb`

then the job limits are reset to ncpus=6 and mem=18gb

5.1.2 Changing the Job-wide Limit

If the job-wide limit is modified, the corresponding resources in the selection directive are not modified. It would be impossible to determine where to apply the changes in a compound directive.

Reducing a job-wide limit to a new value less than the sum of the resource in the directive is strongly discouraged. This may produce a situation where the job is aborted during execution for exceeding its limits. The actual effect of such a modification is not specified.

A job's walltime may be altered at any time, except when the job is in the *Exiting* state, regardless of the initial value.

If a job is queued, requested modifications must still fit within the queue's and server's job resource limits. If a requested modification to a resource would exceed the queue's or server's job resource limits, the resource request will be rejected.

Resources are modified by using the `-l` option, either in chunks inside of selection statements, or in job-wide modifications using `resource_name=value` pairs. The selection statement is of the form:

-l select=[N:]chunk[+[N:]chunk ...]

where N specifies how many of that chunk, and a chunk is of the form:

resource_name=value[:resource_name=value ...]

Job-wide resource_name=value modifications are of the form:

-l resource_name=value[,resource_name=value ...]

It is an error to use a boolean resource as a job-wide limit.

Placement of jobs on vnodes is changed using the place statement:

-l place=modifier[:modifier]

where *modifier* is any combination of *group*, *excl*, *exclhost*, and/or one of *free|pack|scatter|vscatter*.

The usage syntax for `qalter` is:

qalter job-resources job-list

The following examples illustrate how to use the `qalter` command. First we list all the jobs of a particular user. Then we modify two attributes as shown (increasing the wall-clock time from 20 to 25 minutes, and changing the job name from "airfoil" to "engine"):

```
qstat -u barry
                                                  Req'd  Elap
Job ID   User   Queue Jobname Sess NDS TSK Mem Time S Time
-------- ------ ----- ------- ---- --- --- --- ---- - ----
51.south barry  workq airfoil  930  --   1  -- 0:16 R 0:01
54.south barry  workq airfoil  --   --   1  -- 0:20 Q  --

qalter -l walltime=20:00 -N engine 54
qstat -a 54
                                                  Req'd  Elap
Job ID   User   Queue Jobname Sess NDS TSK Mem Time S Time
-------- ------ ----- ------- ---- --- --- --- ---- - ----
54.south barry  workq engine   --   --   1  -- 0:25 Q  --
```

To alter a job attribute via `xpbs`, first select the job(s) of interest, and the click on *modify* button. Doing so will bring up the *Modify Job Attributes* dialog box. From this window you may set the new values for any attribute you are permitted to change. Then click on the *confirm modify* button at the lower left of the window.

The qalter command can be used on job arrays, but not on subjobs or ranges of subjobs. When used with job arrays, any job array identifiers must be enclosed in double quotes, e.g.:

```
qalter -l walltime=25:00 "1234[].south"
```

You cannot use the `qalter` command (or any other command) to alter a custom resource which has been created to be invisible or unrequestable. See section 3.5.15 “Resource Permissions” on page 49.

For more information, see the `qalter(1B)` manual page.

5.2 Holding and Releasing Jobs

PBS provides a pair of commands to hold and release jobs. To hold a job is to mark it as ineligible to run until the hold on the job is “released”.

The **`qhold`** command requests that a Server place one or more holds on a job. A job that has a hold is not eligible for execution. There are three types of holds: *user*, *operator*, and *system*. A user may place a *user* hold upon any job the user owns. An “operator”, who is a user with “operator privilege”, may place either an *user* or an *operator* hold on any job. The PBS Manager may place any hold on any job. The usage syntax of the `qhold` command is:

qhold [-h hold_list] job_identifier ...

Note that for a job array the `job_identifier` must be enclosed in double quotes.

The *`hold_list`* defines the type of holds to be placed on the job. The *`hold_list`* argument is a string consisting of one or more of the letters `u`, `p`, `o`, or `s` in any combination, or the letter `n`. The hold type associated with each letter is:

Table 5-1: Hold Types

Letter	Meaning
n	none - no hold type specified
u	user - the user may set and release this hold type
p	password - set if job fails due to a bad password; can be unset by the user
o	operator; require operator privilege to unset

Table 5-1: Hold Types

Letter	Meaning
s	system - requires manager privilege to unset

If no -h option is given, the *user* hold will be applied to the jobs described by the *job_identifier* operand list. If the job identified by *job_identifier* is in the queued, held, or waiting states, then all that occurs is that the hold type is added to the job. The job is then placed into held state if it resides in an execution queue.

If the job is running, then the following additional action is taken to interrupt the execution of the job. If the job is checkpointable, requesting a hold on a running job will cause (1) the job to be checkpointed, (2) the resources assigned to the job to be released, and (3) the job to be placed in the held state in the execution queue. If the job is not checkpointable, qhold will only set the requested hold attribute. This will have no effect unless the job is requeued with the qrerun command. See section 3.13.13.1 “Checkpointable Jobs” on page 86.

The qhold command can be used on job arrays, but not on subjobs or ranges of subjobs. On job arrays, the qhold command can be applied only in the ‘Q’, ‘B’ or ‘W’ states. This will put the job array in the ‘H’, held, state. If any subjobs are running, they will run to completion. Job arrays cannot be moved in the ‘H’ state if any subjobs are running.

Checkpointing is not supported for job arrays. Even on systems that support checkpointing, no subjobs will be checkpointed -- they will run to completion.

Similarly, the **qrls** command releases a hold on a job. However, the user executing the qrls command must have the necessary privilege to release a given hold. The same rules apply for releasing a hold as exist for setting a hold.

The qrls command can only be used with job array objects, not with subjobs or ranges. The job array will be returned to its pre-hold state, which can be either ‘Q’, ‘B’, or ‘W’.

The usage syntax of the qrls command is:

qrls [-h hold_list] job_identifier ...

For job arrays, the job_identifier must be enclosed in double quotes.

The following examples illustrate how to use both the `qhold` and `qrls` commands. Notice that the state ("S") column shows how the state of the job changes with the use of these two commands.

```
qstat -a 54

                                                  Req'd  Elap
Job ID   User   Queue Jobname Sess NDS TSK Mem Time S Time
-------- ------ ----- ------- ---- --- --- --- ---- - ----
54.south barry  workq  engine  --   --   1  -- 0:20 Q  --

qhold 54
qstat -a 54

                                                  Req'd  Elap
Job ID   User   Queue Jobname Sess NDS TSK Mem Time S Time
-------- ------ ----- ------- ---- --- --- --- ---- - ----
54.south barry  workq  engine  --   --   1  -- 0:20 H  --

qrls -h u 54
qstat -a 54

                                                  Req'd  Elap
Job ID   User   Queue Jobname Sess NDS TSK Mem Time S Time
-------- ------ ----- ------- ---- --- --- --- ---- - ----
54.south barry  workq  engine  --   --   1  -- 0:20 Q  --
```

If you attempted to release a hold on a job which is not on hold, the request will be ignored. If you use the `qrls` command to release a hold on a job that had been previously running, and subsequently checkpointed, the hold will be released, and the job will return to the queued (Q) state (and be eligible to be scheduled to run when resources come available).

To hold (or release) a job using `xpbs`, first select the job(s) of interest, then click the *hold* (or *release*) button.

The `qrls` command does not run the job; it simply releases the hold and makes the job eligible to be run the next time the scheduler selects it.

5.3 Deleting Jobs

PBS provides the `qdel` command for deleting jobs. The `qdel` command deletes jobs in the order in which their job identifiers are presented to the command. A batch job may be deleted by its owner, a PBS operator, or a PBS administrator.

Example:

```
qdel 51
qdel 1234[].server
```

Job array identifiers must be enclosed in double quotes.

Mail is sent for each job deleted unless you specify otherwise. Use the following option to `qdel` to prevent more email than you want from being sent:

```
-Wsuppress_email=<N>
```

`N` must be a non-negative integer. Make `N` the largest number of emails you wish to receive per `qdel` command. PBS will send one email for each deleted job, up to `N`. Note that a job array is one job, so deleting a job array results in one email being sent.

To delete a job using `xpbs`, first select the job(s) of interest, then click the *delete* button.

5.3.1 Deleting Finished and Moved Jobs

5.3.1.1 Deleting Finished Jobs

The `qdel` command does not affect finished jobs, whether this job finished at the local server or at the destination server. If you try to delete a finished job, you will get the following error:

```
qdel: Job <jobid> has finished
```

5.3.1.2 Deleting Moved Jobs

A job that has been moved to another server is either finished or still active, i.e. queued or running. If the moved job is active at the destination server, the `qdel` command deletes the job.

5.4 Sending Messages to Jobs

To send a message to a job is to write a message string into one or more output files of the job. Typically this is done to leave an informative message in the output of the job. Such messages can be written using the `qmsg` command.

IMPORTANT:

A message can only be sent to running jobs.

The usage syntax of the `qmsg` command is:

qmsg [-E][-O] message_string job_identifier

Example:

```
qmsg -O "output file message" 54
qmsg -O "output file message" "1234[].server"
```

Job array identifiers must be enclosed in double quotes.

The `-E` option writes the message into the error file of the specified job(s). The `-O` option writes the message into the output file of the specified job(s). If neither option is specified, the message will be written to the error file of the job.

The first operand, *`message_string`*, is the message to be written. If the string contains blanks, the string must be quoted. If the final character of the string is not a newline, a newline character will be added when written to the job's file. All remaining operands are *`job_identifiers`* which specify the jobs to receive the message string. For example:

```
qmsg -E "hello to my error (.e) file" 55
qmsg -O "hello to my output (.o) file" 55
qmsg "this too will go to my error (.e) file" 55
```

To send a message to a job using `xpbs`, first select the job(s) of interest, then click the *msg* button. Doing so will launch the *Send Message to Job* dialog box. From this window, you may enter the message you wish to send and indicate whether it should be written to the standard output or the standard error file of the job. Click the *Send Message* button to complete the process.

5.5 Sending Signals to Jobs

The **`qsig`** command requests that a signal be sent to executing PBS jobs. The signal is sent to the session leader of the job. Usage syntax of the `qsig` command is:

qsig [-s signal] job_identifier

Job array `job_identifiers` must be enclosed in double quotes.

If the `-s` option is not specified, `SIGTERM` is sent. If the `-s` option is specified, it declares which *`signal`* is sent to the job. The *`signal`* argument is either a signal name, e.g. `SIGKILL`, the signal name without the `SIG` prefix, e.g. `KILL`, or an unsigned signal number, e.g. 9. The signal name `SIGNULL` is allowed; the Server will send the signal 0 to the job which will have no effect. Not all signal names will be recognized by `qsig`. If it doesn't recognize the signal name, try issuing the signal number instead. The request to signal a batch job will be rejected if:

- The user is not authorized to signal the job.
- The job is not in the running state.
- The requested signal is not supported by the execution host.
- The job is exiting.

Two special signal names, "suspend" and "resume", (note, all lower case), are used to suspend and resume jobs. When suspended, a job continues to occupy system resources but is not executing and is not charged for wall-time. Manager or operator privilege is required to suspend or resume a job.

The three examples below all send a signal 9 (SIGKILL) to job 34:

```
qsig -s SIGKILL 34
qsig -s KILL 34
```

IMPORTANT:

On most UNIX systems the command "`kill -l`" (that's 'minus ell') will list all the available signals.

To send a signal to a job using `xpbs`, first select the job(s) of interest, then click the *signal* button. Doing so will launch the *Signal Running Job* dialog box.

From this window, you may click on any of the common signals, or you may enter the signal number or signal name you wish to send to the job. Click the *Signal* button to complete the process.

5.6 Changing Order of Jobs

PBS provides the **`qorder`** command to change the order of two jobs, within or across queues. To order two jobs is to exchange the jobs' positions in the queue or queues in which the jobs reside. If job1 is at position 3 in queue A and job2 is at position 4 in queue B, qordering them will result in job1 being in position 4 in queue B and job2 being in position 3 in queue A. The two jobs must be located at the same Server, and both jobs must be owned by the user. No attribute of the job (such as priority) is changed. The impact of changing the order within the queue(s) is dependent on local job scheduling policy; contact your systems administrator for details.

IMPORTANT:

A job in the running state cannot be reordered.

Usage of the `qorder` command is:

qorder job_identifier1 job_identifier2

Job array identifiers must be enclosed in double quotes.

Both operands are *`job_identifiers`* which specify the jobs to be exchanged.

```
qstat -u bob

                                                  Req'd  Elap
Job ID   User   Queue Jobname Sess NDS TSK Mem Time S Time
```

```
-------- ------ ----- ------- ---- --- --- --- ---- - ----
54.south bob    workq twinkie  --   --   1  -- 0:20 Q  --
63[].south bob  workq airfoil  --   --   1  -- 0:13 Q  --
```

```
qorder 54 "63[]"
qstat -u bob
                                                  Req'd     Elap
Job ID   User   Queue Jobname Sess NDS TSK Mem Time S Time
-------- ------ ----- ------- ---- --- --- --- ---- - ----
63[].south bob  workq airfoil  --   --   1  -- 0:13 Q  --
54.south bob    workq twinkie  --   --   1  -- 0:20 Q  --
```

To change the order of two jobs using `xpbs`, select the two jobs, and then click the *order* button.

The qorder command can only be used with job array objects, not on subjobs or ranges. This will change the queue order of the job array in association with other jobs or job arrays in the queue.

5.7 Moving Jobs Between Queues

PBS provides the **`qmove`** command to move jobs between different queues (even queues on different Servers). To move a job is to remove the job from the queue in which it resides and instantiate the job in another queue.

IMPORTANT:

A job in the running state cannot be moved.

The usage syntax of the `qmove` command is:

qmove destination job_identifier(s)

Job array `job_identifiers` must be enclosed in double quotes.

The first operand is the new destination for

```
queue
@server
queue@server
```

If the *destination* operand describes only a queue, then qmove will move jobs into the queue of the specified name at the job's current Server. If the *destination* operand describes only a Server, then qmove will move jobs into the default queue at that Server. If the *destination* operand describes both a queue and a Server, then qmove will move the jobs into the specified queue at the specified Server. All following operands are *job_identifiers* which specify the jobs to be moved to the new *destination*.

To move jobs between queues or between Servers using xpbs, select the job(s) of interest, and then click the move button. Doing so will launch the Move Job dialog box from which you can select the queue and/or Server to which you want the job(s) moved.

The qmove command can only be used with job array objects, not with subjobs or ranges. Job arrays can only be moved from one server to another if they are in the 'Q', 'H', or 'W' states, and only if there are no running subjobs. The state of the job array object is preserved in the move. The job array will run to completion on the new server.

As with jobs, a qstat on the server from which the job array was moved will not show the job array. A qstat on the job array object will be redirected to the new server.

Note: The subjob accounting records will be split between the two servers.

5.8 Converting a Job into a Reservation Job

The pbs_rsub command can be used to convert a normal job into a reservation job that will run as soon as possible. PBS creates a reservation queue and a reservation, and moves the job into the queue. Other jobs can also be moved into that queue via qmove(1B) or submitted to that queue via qsub(1B). The reservation is called an ASAP reservation.

The format for converting a normal job into a reservation job is:

pbs_rsub [-l walltime=time] -W qmove=job_identifier

Example:

```
pbs_rsub -W qmove=54
pbs_rsub -W qmove="1234[].server"
```

The -R and -E options to pbs_rsub are disabled when using the -W qmove option.

For more information, see "Advance and Standing Reservation of Resources" on page 181, and the pbs_rsub(1B), qsub(1B) and qmove(1B) manual pages.

A job's default walltime is 5 years. Therefore an ASAP reservation's start time can be in 5 years, if all the jobs in the system have the default walltime.

You cannot use the pbs_rsub command (or any other command) to request a custom resource which has been created to be invisible or unrequestable. See section 3.5.15 "Resource Permissions" on page 49.

5.9 Using Job History Information

5.9.1 Introduction

PBS Professional can provide job history information, including what the submission parameters were, whether the job started execution, whether execution succeeded, whether staging out of results succeeded, and which resources were used.

PBS can keep job history for jobs which have finished execution, were deleted, or were moved to another server.

5.9.2 Definitions

Moved jobs

Jobs which were moved to another server

Finished jobs

Jobs whose execution is done, for any reason:

- Jobs which finished execution successfully and exited
- Jobs terminated by PBS while running
- Jobs whose execution failed because of system or network failure
- Jobs which were deleted before they could start execution

5.9.3 Job History Information

PBS can keep all job attribute information, including the following:

- Submission parameters
- Whether the job started execution
- Whether execution succeeded
- Whether staging out of results succeeded
- Which resources were used

PBS keeps job history for the following jobs:

- Jobs that have finished execution
- Jobs that were deleted
- Jobs that were moved to another server

The job history for finished and moved jobs is preserved and available for the specified duration. After the duration has expired, PBS deletes the job history information and it is no longer available. The state of a finished job is *F*, and the state of a moved job is *M*. See "Job States" on page 463 of the PBS Professional Reference Guide.

Subjobs are not considered finished jobs until the parent array job is finished, which happens when all of its subjobs have terminated execution.

5.9.4 Working With Finished and Moved Jobs

5.9.4.1 Working With Moved Jobs

You can use the following commands with moved jobs. They will function as they do with normal jobs.

```
qdel
qalter
qhold
qmove
qmsg
qorder
qrerun
qrls
qrun
qsig
```

5.9.4.2 PBS Commands and Finished Jobs

The commands listed above cannot be used with finished jobs, whether they finished at the local server or a remote server. These jobs are no longer running; PBS is storing their information, and this information cannot be deleted, altered, etc. Trying to use one of the above commands with a finished job results in the following error message:

```
<command name>: Job <jobid> has finished
```

5.9.5 Viewing Information for Finished and Moved Jobs

You can view information for finished and moved jobs in the same way as for queued and running jobs, as long as the job history is still being preserved.

The -x option to the `qstat` command allows you to see information for all jobs, whether they are running, queued, finished or moved. This information is presented in standard format. The -H option to the `qstat` command allows you to see alternate-format information for finished or moved jobs only. See section 6.1.20 “Viewing Job History” on page 149.

5.9.5.1 UNIX/Linux:

```
qstat -fx `qselect -x -s “MF”`
```

5.9.5.2 Windows:

```
for /F "usebackq" %%j in (`"\Program Files\ PBSPro\
   exec\ bin\qselect" -x -s MF`)
    do ("\Program Files\PBS Pro\exec\bin\qstat" -fx
   %%j)
```

5.9.6 Listing Job Identifiers of Finished and Moved Jobs

You can list identifiers of finished and moved jobs in the same way as for queued and running jobs, as long as the job history is still being preserved.

The -x option to the `qselect` command allows you to list job identifiers for all jobs, whether they are running, queued, finished or moved. The -H option to the `qselect` command allows you to list job identifiers for finished or moved jobs only. See section 6.3 “The qselect Command” on page 154.

5.9.6.1 Listing Jobs by Time Attributes

You can use the `qselect` command to list queued, running, finished and moved jobs, job arrays, and subjobs according to their time attributes. The -t option to the qselect command allows you to specify how you want to select based on time attributes. You can also use the -t option twice to bracket a time period. See section 6.3 “The qselect Command” on page 154.

Example 1: Select jobs with end time between noon and 3PM.

```
qselect -te.gt.09251200 -te.lt.09251500
```

Example 2: Select finished and moved jobs with start time between noon and 3PM.

```
qselect -x -s “MF” -ts.gt.09251200 -ts.lt.09251500
```

Example 3: Select all jobs with creation time between noon and 3PM

```
qselect -x -tc.gt.09251200 -tc.lt.09251500
```

Example 4: Select all jobs including finished and moved jobs with qtime of 2.30PM (default relation is .eq.)

```
qselect -x -tq09251430
```

Chapter 6

Checking Job / System Status

This chapter introduces several PBS commands useful for checking status of jobs, queues, and PBS Servers. Examples for use are included, as are instructions on how to accomplish the same task using the `xpbs` graphical interface.

6.1 The `qstat` Command

The **`qstat`** command is used to the request the status of jobs, queues, and the PBS Server. The requested status is written to standard output stream (usually the user's terminal). When requesting job status, any jobs for which the user does not have view privilege are not displayed. For detailed usage information, see "qstat" on page 228 of the PBS Professional Reference Guide.

Usage:

```
qstat [-J] [-p] [-t] [-x] [[ job_identifier | destina-
   tion ] ...]
qstat -f [-J] [-p] [-t] [-x] [[ job_identifier | desti-
   nation ] ...]
qstat [-a [-w] | -H | -i | -r ] [-G|-M] [-J] [-n [-1][-
   w]] [-s [-1][-w]] [-t] [-T [-w]] [-u user] [[job_id
   | destination] ...]
qstat -Q [-f] [ destination... ]
qstat -q [-G|-M] [ destination... ]
qstat -B [-f] [ server_name... ]
qstat --version
```

6.1.1 Checking Job Status

Executing the `qstat` command without any options displays job information in the default format. (An alternative display format is also provided, and is discussed below.) The default display includes the following information:

- The job identifier assigned by PBS
- The job name given by the submitter
- The job owner
- The CPU time used
- The job state
- The queue in which the job resides

See “Job States” on page 463 of the PBS Professional Reference Guide.

The following example illustrates the default display of `qstat`.

```
qstat
Job id    Name         User         Time Use S Queue
--------- ---------- ---------- -------- - -----
16.south  aims14      user1              0 H workq
18.south  aims14      user1              0 W workq
26.south  airfoil     barry       00:21:03 R workq
27.south  airfoil     barry       21:09:12 R workq
28.south  myjob       user1              0 Q workq
29.south  tns3d       susan              0 Q workq
30.south  airfoil     barry              0 Q workq
31.south  seq_35_3    donald             0 Q workq
```

An alternative display (accessed via the "`-a`" option) is also provided that includes extra information about jobs, including the following additional fields:

Session ID

Number of vnodes requested

Number of parallel tasks (or CPUs)

Requested amount of memory

Requested amount of wall clock time

Walltime or CPU time, whichever submitter specified, if job is running.

```
qstat -a
                                                  Req'd  Elap
Job ID   User   Queue Jobname Ses NDS TSK Mem Time S Time
-------- ------ ----- ------- --- --- --- --- ---- - ----
16.south user1  workq aims14  --  --   1  --  0:01 H  --
18.south user1  workq aims14  --  --   1  --  0:01 W  --
51.south barry  workq airfoil 930 --   1  --  0:13 R 0:01
52.south user1  workq myjob   --  --   1  --  0:10 Q  --
53.south susan  workq tns3d   --  --   1  --  0:20 Q  --
54.south barry  workq airfoil --  --   1  --  0:13 Q  --
```

```
55.south donald workq seq_35_ --  --   1  --  2:00 Q  --
```

Other options which utilize the alternative display are discussed in subsequent sections of this chapter.

6.1.2 Viewing Specific Information

When requesting queue or Server status `qstat` will output information about each destination. The various options to `qstat` take as an operand either a job identifier or a destination. If the operand is a job identifier, it must be in the following form:

sequence_number[.server_name][@server]

where `sequence_number.server_name` is the job identifier assigned at submittal time, see `qsub`. If the `.server_name` is omitted, the name of the default Server will be used. If `@server` is supplied, the request will be for the job identifier currently at that Server.

If the operand is a destination identifier, it takes one of the following three forms:

queue

@server

queue@server

If *`queue`* is specified, the request is for status of all jobs in that queue at the default Server. If the *`@server`* form is given, the request is for status of all jobs at that Server. If a full destination identifier, *`queue@server`*, is given, the request is for status of all jobs in the named *`queue`* at the named *`server`*.

IMPORTANT:

If a PBS Server is not specified on the `qstat` command line, the default Server will be used. (See discussion of **`PBS_DEFAULT`** in section 2.15 “Environment Variables” on page 20.)

6.1.3 Checking Server Status

The "**-B**" option to `qstat` displays the status of the specified PBS Batch Server. One line of output is generated for each Server queried. The three letter abbreviations correspond to various job limits and counts as follows: Maximum, Total, Queued, Running, Held, Waiting, Transiting, and Exiting. The last column gives the status of the Server itself: active, idle, or scheduling.

```
qstat -B
Server      Max  Tot  Que  Run  Hld  Wat  Trn  Ext Status
----------- --- ---- ---- ---- ---- ---- ---- ---- ------
fast.domain   0   14   13    1    0    0    0    0 Active
```

When querying jobs, Servers, or queues, you can add the "`-f`" option to `qstat` to change the display to the *full* or *long* display. For example, the Server status shown above would be expanded using "`-f`" as shown below:

```
qstat -Bf
Server: fast.mydomain.com
   server_state = Active
   scheduling = True
   total_jobs = 14
   state_count = Transit:0 Queued:13 Held:0 Waiting:0
      Running:1 Exiting:0
   managers = user1@fast.mydomain.com
   default_queue = workq
   log_events = 511
   mail_from = adm
   query_other_jobs = True
   resources_available.mem = 64mb
   resources_available.ncpus = 2
   resources_default.ncpus = 1
   resources_assigned.ncpus = 1
   resources_assigned.nodect = 1
   scheduler_iteration = 600
   pbs_version = PBSPro_11.1.41640
```

6.1.4 Checking Queue Status

The "**`-Q`**" option to `qstat` displays the status of all (or any specified) queues at the (optionally specified) PBS Server. One line of output is generated for each queue queried. The three letter abbreviations correspond to

limits, queue states, and job counts as follows: Maximum, Total, Enabled Status, Started Status, Queued, Running, Held, Waiting, Transiting, and Exiting. The last column gives the type of the queue: *routing* or *execution*.

```
qstat -Q
Queue Max Tot Ena Str Que Run Hld Wat Trn Ext Type
----- --- --- --- --- --- --- --- --- --- --- ---------
workq   0  10 yes yes   7   1   1   1   0   0 Execution
```

The full display for a queue provides additional information:

```
 qstat -Qf
Queue: workq
   queue_type = Execution
   total_jobs = 10
   state_count = Transit:0 Queued:7 Held:1 Waiting:1
      Running:1 Exiting:0
   resources_assigned.ncpus = 1
   hasnodes = False
   enabled = True
   started = True
```

6.1.5 Viewing Job Information

We saw above that the "`-f`" option could be used to display full or long information for queues and Servers. The same applies to jobs. By specifying the "`-f`" option and a job identifier, PBS will print all information known about the job (e.g. resources requested, resource limits, owner,

source, destination, queue, etc.) as shown in the following example. (See "Job Attributes" on page 430 of the PBS Professional Reference Guide for a description of attributes.)

```
qstat -f 13
Job Id: 13.host1
    Job_Name = STDIN
    Job_Owner = user1@host2
    resources_used.cpupercent = 0
    resources_used.cput = 00:00:00
    resources_used.mem = 2408kb
    resources_used.ncpus = 1
    resources_used.vmem = 12392kb
    resources_used.walltime = 00:01:31
    job_state = R
    queue = workq
    server = host1
    Checkpoint = u
    ctime = Thu Apr  2 12:07:05 2010
    Error_Path = host2:/home/user1/STDIN.e13
    exec_host = host2/0
    exec_vnode = (host3:ncpus=1)
    Hold_Types = n
    Join_Path = n
    Keep_Files = n
    Mail_Points = a
    mtime = Thu Apr  2 12:07:07 2010
    Output_Path = host2:/home/user1/STDIN.o13
    Priority = 0
    qtime = Thu Apr  2 12:07:05 2010
    Rerunable = True
    Resource_List.ncpus = 1
    Resource_List.nodect = 1
```

```
 Resource_List.place = free
 Resource_List.select = host=host3
 stime = Thu Apr  2 12:07:08 2010
 session_id = 32704
 jobdir = /home/user1
 substate = 42
 Variable_List = PBS_O_HOME=/home/
user1,PBS_O_LANG=en_US.UTF-8,
     PBS_O_LOGNAME=user1,
     PBS_O_PATH=/opt/gnome/sbin:/root/bin:/usr/
local/bin:/usr/bin:/usr/X11R
     6/bin:/bin:/usr/games:/opt/gnome/bin:/opt/kde3/
bin:/usr/lib/mit/bin:/us
     r/lib/mit/sbin,PBS_O_MAIL=/var/mail/
root,PBS_O_SHELL=/bin/bash,
     PBS_O_HOST=host2,PBS_O_WORKDIR=/home/
user1,PBS_O_SYSTEM=Linux,
     PBS_O_QUEUE=workq
 comment = Job run at Thu Apr 02 at 12:07 on
(host3:ncpus=1)
 alt_id = <dom0:JobID xmlns:dom0="http://sche-
mas.microsoft.com/HPCS2008/hpcb
     p">149</dom0:JobID>
 etime = Thu Apr  2 12:07:05 2010
 Submit_arguments = -lselect=host=host3 -- ping -n
100 127.0.0.1
 executable = <jsdl-hpcpa:Executable>ping</jsdl-
hpcpa:Executable>
 argument_list = <jsdl-hpcpa:Argument>-n</jsdl-
hpcpa:Argument><jsdl-hpcpa:Ar
     gument>100</jsdl-hpcpa:Argument><jsdl-
hpcpa:Argument>127.0.0.1</jsdl-hp
     cpa:Argument>
```

6.1.6 List User-Specific Jobs

The "**-u**" option to `qstat` displays jobs owned by any of a list of user names specified. The syntax of the list of users is:

user_name[@host][,user_name[@host],...]

Host names are not required, and may be "wild carded" on the left end, e.g. "`*.mydomain.com`". *`user_name`* without a "`@host`" is equivalent to "`user_name@*`", that is at any host.

`qstat -u user1`

```
                                                  Req'd  Elap
Job ID   User   Queue Jobname Sess NDS TSK Mem Time S Time
-------- ------ ----- ------- ---- --- --- --- ---- - ----
16.south user1  workq aims14   --   --   1  -- 0:01 H  --
18.south user1  workq aims14   --   --   1  -- 0:01 W  --
52.south user1  workq my_job   --   --   1  -- 0:10 Q  --
```

`qstat -u user1,barry`

```
51.south barry  workq airfoil  930  --   1  -- 0:13 R 0:01
52.south user1  workq my_job   --   --   1  -- 0:10 Q  --
54.south barry  workq airfoil  --   --   1  -- 0:13 Q  --
```

6.1.7 List Running Jobs

The "**-r**" option to `qstat` displays the status of all running jobs at the (optionally specified) PBS Server. Running jobs include those that are running and suspended. One line of output is generated for each job reported, and the information is presented in the alternative display. For example:

`qstat –r`

```
host1:
                                                  Req'd  Req'd     Elap
Job ID      User  Queue Jobname Sess NDS TSK  Mem    Time   S  Time
----------- ----- ----- ------- ---- --- ---  -----  -----  -  -----
43.host1    user1 workq STDIN   4693   1   1     --     --  R  00:00
```

6.1.8 List Non-Running Jobs

The "**-i**" option to `qstat` displays the status of all non-running jobs at the (optionally specified) PBS Server. Non-running jobs include those that are queued, held, and waiting. One line of output is generated for each job reported, and the information is presented in the alternative display (see description above). For example:

```
qstat –i
host1:
                                                  Req'd  Req'd     Elap
Job ID      User  Queue Jobname Sess NDS TSK  Mem    Time   S  Time
----------  ----- ----- ------- ---- --- ---  -----  -----  -  -----
44[].host1  user1 workq STDIN     --   1   1     --     --  Q     --
```

6.1.9 Display Size in Gigabytes

The "**-G**" option to `qstat` displays all jobs at the requested (or default) Server using the alternative display, showing all size information in gigabytes (GB) rather than the default of smallest displayable units. Note that if the size specified is less than 1 GB, then the amount if rounded up to 1 GB. For example:

```
qstat –G
host1:
                                                  Req'd  Req'd     Elap
Job ID     User  Queue Jobname Sess NDS TSK  Mem    Time   S  Time
---------  ----- ----- ------- ---- --- ---  -----  -----  -  -----
43.host1   user1 workq STDIN   4693   1   1     --     --  R  00:05
44[].host1 user1 workq STDIN     --   1   1     --     --  Q     --
45.host1   user1 workq STDIN     --   1   1    1gb     --  Q     --
```

6.1.10 Display Size in Megawords

The "**-M**" option to `qstat` displays all jobs at the requested (or default) Server using the alternative display, showing all size information in megawords (MW) rather than the default of smallest displayable units. A word is considered to be 8 bytes. For example:

qstat –M

```
host1:
                                                  Req'd  Req'd     Elap
Job ID     User  Queue Jobname Sess NDS TSK  Mem    Time   S  Time
---------- ----- ----- ------- ---- --- ---  -----  -----  -  -----
43.host1   user1 workq STDIN   4693   1   1     --     --  R  00:05
44[].host1 user1 workq STDIN     --   1   1     --     --  Q     --
45.host1   user1 workq STDIN     --   1   1   25mw     --  Q     --
```

6.1.11 List Hosts Assigned to Jobs

The "**-n**" option to `qstat` displays the hosts allocated to any running job at the (optionally specified) PBS Server, in addition to the other information presented in the alternative display. The host information is printed immediately below the job (see job 51 in the example below), and includes the host name and number of virtual processors assigned to the job (i.e. "`south/0`", where "`south`" is the host name, followed by the virtual processor(s) assigned.). A text string of "--" is printed for non-running jobs. Notice the differences between the queued and running jobs in the example below:

qstat -n

```
                                                    Req'd  Elap
Job ID   User   Queue Jobname Sess NDS TSK Mem Time S Time
-------- ------ ----- ------- ---- --- --- --- ---- - ----
16.south user1  workq aims14   --   --   1  -- 0:01 H  --
    --
18.south user1  workq aims14   --   --   1  -- 0:01 W  --
    --
51.south barry  workq airfoil  930  --   1  -- 0:13 R
```

```
    0:01 south/0
52.south user1  workq my_job   --   --   1  -- 0:10 Q  --
    --
```

6.1.12 Display Job Comments

The "**-s**" option to `qstat` displays the job comments, in addition to the other information presented in the alternative display. The job comment is printed immediately below the job. By default the job comment is updated by the Scheduler with the reason why a given job is not running, or when the job began executing. A text string of "--" is printed for jobs whose comment has not yet been set. The example below illustrates the different type of messages that may be displayed:

```
qstat -s
                                                  Req'd  Elap
Job ID   User  Queue Jobname Sess NDS TSK Mem Time S Time
-------- ----- ----- ------- ---- --- --- --- ---- - ----
16.south user1 workq aims14   --   --   1  -- 0:01 H  --
    Job held by user1 on Wed Aug 22 13:06:11 2004
18.south user1 workq aims14   --   --   1  -- 0:01 W  --
    Waiting on user requested start time
51.south barry workq airfoil  930  --   1  -- 0:13 R 0:01
    Job run on host south - started Thu Aug 23 at 10:56
52.south user1 workq my_job   --   --   1  -- 0:10 Q  --
    Not Running: No available resources on nodes
57.south susan workq solver   --   --   2  -- 0:20 Q  --
    --
```

6.1.13 Display Queue Limits

The "**-q**" option to `qstat` displays any limits set on the requested (or default) queues. Since PBS is shipped with no queue limits set, any visible limits will be site-specific. The limits are listed in the format shown below.

```
qstat -q
server: south
```

```
Queue  Memory CPU Time Walltime Node Run Que Lm  State
------ ------ -------- -------- ---- --- --- --  -----
workq    --      --       --     --    1   8 --   E R
```

6.1.14 Show State of Job, Job Array or Subjob

The “-t” option to `qstat` will show the state of a job, a job array object, and all non-X subjobs.

The “-J” option to `qstat` will show only the state of job arrays.

The combination of “-J” and “-t” options to `qstat` will show only the state of subjobs.

For example:

`qstat –t`

```
Job ID        Name      User         Time Use  S  Queue
----------    --------  -----------  --------  -  -----
44[].host1    STDIN     user1               0  B  workq
44[1].host1   STDIN     user1        00:00:00  R  workq
44[2].host1   STDIN     user1               0  Q  workq
44[3].host1   STDIN     user1               0  Q  workq
```

`qstat –J`

```
Job ID        Name      User         Time Use  S  Queue
-----------   --------  -----------  --------  -  -----
44[].host1    STDIN     user1               0  B  workq
```

`$ qstat –Jt`

```
Job ID         Name      User         Time Use  S  Queue
-----------    --------  -----------  --------  -  -----
44[1].host1   STDIN     user1        00:00:00  R  workq
44[2].host1   STDIN     user1               0  Q  workq
44[3].host1   STDIN     user1               0  Q  workq
```

6.1.15 Viewing Job Start Time

There are two ways you can find the job's start time. If the job is still running, you can do a `qstat -f` and look for the stime attribute. If the job has finished, you look in the accounting log for the *S* record for the job. For an array job, only the *S* record is available.

6.1.16 Viewing Job Status in Wide Format

The `–w qstat` option displays job status in wide format. The total width of the display is extended from 80 characters to 120 characters. The Job ID column can be up to 30 characters wide, while the Username, Queue, and Jobname column can be up to 15 characters wide. The SessID column can be up to eight characters wide, and the NDS column can be up to four characters wide.

Note: You can use this option only with the `-a`, `-n`, or `-s qstat` options.

6.1.17 Print Job Array Percentage Completed

The "-p" option to `qstat` prints the default display, with a column for Percentage Completed. For a job array, this is the number of subjobs completed and deleted, divided by the total number of subjobs. For example:

```
qstat –p
Job ID          Name      User           % done  S  Queue
-------------   --------  -----------  --------  -  -----
44[].host1      STDIN     user1              40  B  workq
```

6.1.18 Getting Information on Jobs Moved to Another Server

If your job is running at another server, you can use the `qstat` command to see its status. If your site is using peer scheduling, your job may be moved to a server that is not your default server. When that happens, you will need to give the job ID as an argument to `qstat`. If you use only

"qstat", your job will not appear to exist. For example: you submit a job to ServerA, and it returns the jobid as "123.ServerA". Then 123.ServerA is moved to ServerB. In this case, use

```
qstat 123
```

or

```
qstat 123.ServerA
```

to get information about your job. ServerA will query ServerB for the information. To list all jobs at ServerB, you can use:

```
qstat @ServerB
```

If you use "qstat" without the job ID, the job will not appear to exist.

6.1.19 Viewing Resources Allocated to a Job

The exec_vnode attribute displayed via qstat shows the resources allocated from each vnode for the job.

The exec_vnode line looks like:

```
exec_vnode = (<vnode name>:ncpus=W:mem=X)+(<vnode
   name>:ncpus=Y:mem=Z)
```

For example, a job requesting

```
-l select=2:ncpus=1:mem=1gb+1:ncpus=4:mem=2gb
```

gets an exec_vnode of

```
exec_vnode =
   (VNA:ncpus=1:mem=1gb)+(VNB:ncpus=1:mem=1gb)
   +(VNC:ncpus=4:mem=2gb)
```

Note that the vnodes and resources required to satisfy a chunk are grouped by parentheses. In the example above, if two vnodes on a single host were required to satisfy the last chunk, the exec_vnode might be:

```
exec_vnode =
   (VNA:ncpus=1:mem=1gb)+(VNB:ncpus=1:mem=1gb)
   +(VNC1:ncpus=2:mem=1gb+VNC2:ncpus=2:mem=1gb)
```

Note also that if a vnode is allocated to a job because the job requests an arrangement of *exclhost*, only the vnode name appears in the chunk. For example, if a job requesting

```
-l select 2:ncpus=4 -l place = exclhost
```

is placed on a host with 4 vnodes, each with 4 CPUs, the exec_vnode attribute looks like this:

```
exec_vnode = (VN0:ncpus=4)+(VN1:ncpus=4)+(VN2)+(VN3)
```

You cannot use the `qstat` command to view any custom resource which has been created to be invisible or unrequestable, whether this resource is on a queue, the server, or is a job attribute. See section 3.5.15 “Resource Permissions” on page 49.

6.1.19.1 Resources for Requeued Jobs

When a job is requeued due to an error in the prologue or initialization, the job’s exec_host and exec_vnode attributes are cleared. The only exception is when the job is checkpointed and must be rerun on the exact same system. In this case, the exec_host and exec_vnode attributes are preserved.

6.1.20 Viewing Job History

You can view information for jobs that have finished or were moved, as long as that information is still being stored by PBS. See section 5.9 “Using Job History Information” on page 129.

You can view the same attribute information regardless of whether the job is queued, running, finished, or moved, as long as job history information is being preserved.

6.1.20.1 Job History In Standard Format

You can use the -x option to the `qstat` command to see information for finished, moved, queued, and running jobs, in standard format.

Usage:

qstat -x

Displays information for queued, running, finished, and moved jobs, in standard format.

qstat -x <job ID>

Displays information for a job, regardless of its state, in standard format.

Example 1: Showing finished and moved jobs with queued and running jobs:

```
qstat -x
Job id            Name         User    Time Use  S   Queue
------------      -----------  ------  -------  ---  ------
101.server1       STDIN        user1   00:00:00  F   workq
102.server1       STDIN        user1   00:00:00  M   destq@server2
103.server1       STDIN        user1   00:00:00  R   workq
104.server1       STDIN        user1   00:00:00  Q   workq
```

To see status for jobs, job arrays and subjobs that are queued, running, finished, and moved, use `qstat -xt`.

To see status for job arrays that are queued, running, finished, or moved, use `qstat -xJ`.

When information for a moved job is displayed, the destination queue and server are shown as <queue>@<server>.

Example 2: `qstat -x` output for moved job: destination queue is `destq`, and destination server is `server2`.

```
Job id            Name         User    Time Use  S   Queue
----------------  -----------  -----   -------  ---  ------
101.sequoia       STDIN        user1   00:00:00  F   workq
102.sequoia       STDIN        user1   00:00:00  M   destq@server2
103.sequoia       STDIN        user1   00:00:00  R   workq
```

Example 3: Viewing moved job:

- There are three servers with hostnames ServerA, ServerB, and

ServerC

- User1 submits job 123 to ServerA.
- After some time, User1 moves the job to ServerB.
- After more time, the administrator moves the job to QueueC at ServerC.

This means:

- The `qstat` command will show QueueC@ServerC for job 123.

6.1.20.2 Job History In Alternate Format

You can use the -H option to the `qstat` command to see job history for finished or moved jobs in alternate format.

Usage:

qstat -H

Displays information for finished or moved jobs, in alternate format

qstat -H job identifier

Displays information for that job in alternate format, whether or not it is finished or moved

qstat -H destination

Displays information for finished or moved jobs at that destination

Example 1: Job history in alternate format:

```
qstat -H
                                                  Req'd  Req'd   Elap
Job ID  Username  Queue Jobname  SessID  NDS TSK Memory Time  S Time
------  --------  ----  -------  ------  --- --- ------ ----  -- -----
101.S1  user1     workq STDIN    5168     1  1   --     --    F  00:00
102.S1  user1     Q1@S2 STDIN     --      1  2   --     --    M   --
```

To see alternate-format status for jobs, job arrays and subjobs that are finished and moved, use `qstat -Ht`.

To see alternate-format status for job arrays that are finished or moved, use `qstat -HJ`.

The -H option is incompatible with the -a, -i and -r options.

6.1.21 Viewing Estimated Start Times For Jobs

You can view the estimated start times and vnodes of jobs using the `qstat` command. If you use the -T option to `qstat` when viewing job information, the *Elap Time* field is replaced with the *Est Start* field. Running jobs are shown above queued jobs.

See "qstat" on page 228 of the PBS Professional Reference Guide.

If the estimated start time or vnode information is invisible to unprivileged users, no estimated start time or vnode information is available via `qstat`.

Example output:

```
qstat -T

                                                        Req'd  Req'd   Est
Job ID   Username Queue Jobname SessID NDS TSK Memory Time  S Start
-------  -------- ----- -------- ----- --- --- ------ ----- - -----
5.host1   user1   workq  foojob  12345  1   1  128mb 00:10  R   --
9.host1   user1   workq  foojob   --    1   1  128mb 00:10  Q  11:30
10.host1  user1   workq  foojob   --    1   1  128mb 00:10  Q  Tu 15
7.host1   user1   workq  foojob   --    1   1  128mb 00:10  Q  Jul
8.host1   user1   workq  foojob   --    1   1  128mb 00:10  Q  2010
11.host1  user1   workq  foojob   --    1   1  128mb 00:10  Q  >5yrs
13.host1  user1   workq  foojob   --    1   1  128mb 00:10  Q   --
```

If the start time for a job cannot be estimated, the start time behaves as if it is unset. For `qstat -T`, the start time appears as a question mark ("?"). for `qstat -f`, the start time appears as a time in the past.

6.1.22 Caveats

- MOM periodically polls jobs for usage by the jobs running on her host, collects the results, and reports this to the server. When a job exits, she

polls again to get the final tally of usage for that job.

For example, MOM polls the running jobs at times T1, T2, T4, T8, T16, T24, and so on.

The output shown by a `qstat` during the window of time between T8 and T16 shows the resource usage up to T8.

If the `qstat` is done at T17, the output shows usage up through T16. If the job ends at T20, the accounting log (and the final log message, and the email to the user if "`qsub -me`" was used in job submission) contains usage through T20.

- The final report does not include the epilogue. The time required for the epilogue is treated as system overhead.
- The order in which jobs are displayed is undefined.

6.2 Viewing Job / System Status with xpbs

The main display of `xpbs` shows a brief listing of all selected Servers, all queues on those Servers, and any jobs in those queues that match the *selection criteria* (discussed below). Servers are listed in the HOST panel near the top of the display.

To view detailed information about a given Server (i.e. similar to that produced by "`qstat -fB`") select the Server in question, then click the "Detail" button. Likewise, for details on a given queue (i.e. similar to that produced by "`qstat -fQ`") select the queue in question, then click its corresponding "Detail" button. The same applies for jobs as well (i.e. "`qstat -f`"). You can view detailed information on any displayed job by selecting it, and then clicking on the "Detail" button. Note that the list of jobs displayed will be dependent upon the Selection Criteria currently selected. This is discussed in the `xpbs` portion of the next section.

6.3 The qselect Command

The **qselect** command provides a method to list the job identifier of those jobs, job arrays or subjobs which meet a list of selection criteria. Jobs are selected from those owned by a single Server. When qselect successfully completes, it will have written to standard output a list of zero or more job identifiers which meet the criteria specified by the options. Each option acts as a filter restricting the number of jobs which might be listed. With no options, the qselect command will list all jobs at the Server which the user is authorized to list (query status of). The -u option may be used to limit the selection to jobs owned by this user or other specified users.

For a description of the qselect command, see “qselect” on page 215 of the PBS Professional Reference Guide.

For example, say you want to list all jobs owned by user “barry” that requested more than 16 CPUs. You could use the following qselect command syntax:

```
qselect -u barry -l ncpus.gt.16
121.south
133.south
154.south
```

Notice that what is returned is the job identifiers of jobs that match the selection criteria. This may or may not be enough information for your purposes. Many users will use shell syntax to pass the list of job identifiers directly into qstat for viewing purposes, as shown in the next example (necessarily different between UNIX and Windows).

UNIX:

```
qstat -a  ‘ qselect -u barry -l ncpus.gt.16 ‘
                                                  Req'd     Elap
Job ID    User  Queue Jobname Sess NDS TSK Mem Time S Time
--------  ----- ----- ------- ---- --- --- --- ---- - ----
121.south barry workq airfoil  --   --  32  -- 0:01 H   --
133.south barry workq trialx   --   --  20  -- 0:01 W   --
154.south barry workq airfoil 930   --  32  -- 1:30 R 0:32
```

Windows (type the following at the `cmd` prompt, all on one line):

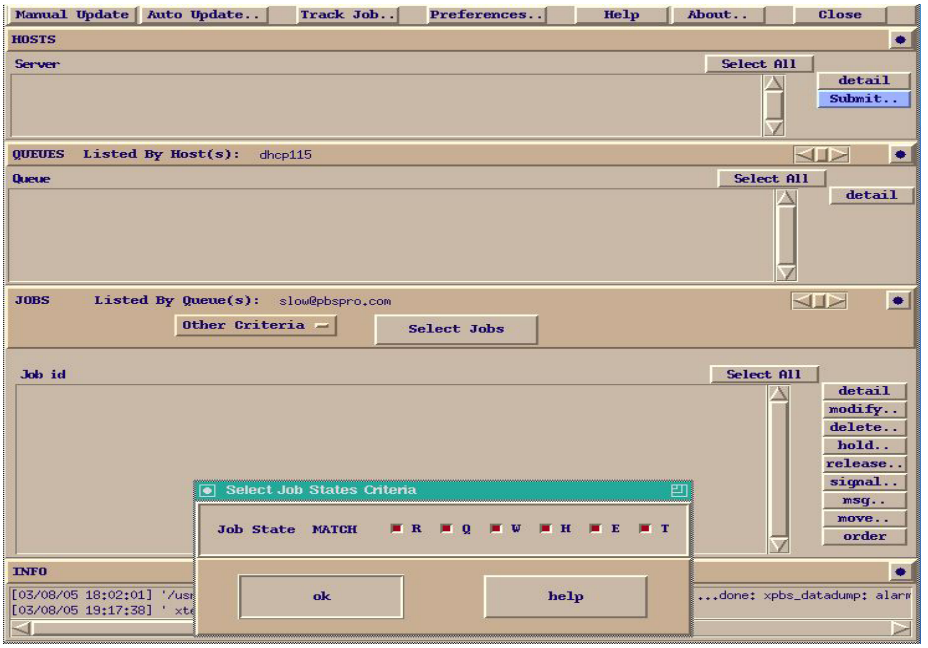
```
for /F "usebackq" %j in (`qselect -u barry -l
   ncpus.gt.16`)  do
( qstat -a %j )
121.south
133.south
154.south
```

Note: This technique of using the output of the `qselect` command as input to `qstat` can also be used to supply input to other PBS commands as well.

6.4 Selecting Jobs Using xpbs

The `xpbs` command provides a graphical means of specifying job selection criteria, offering the flexibility of the `qselect` command in a point and click interface. Above the JOBS panel in the main `xpbs` display is the *Other Criteria* button. Clicking it will bring up a menu that lets you choose and select any job selection criteria you wish.

The example below shows a user clicking on the *Other Criteria* button, then selecting *Job States*, to reveal that all job states are currently selected. Clicking on any of these job states would remove that state from the selection criteria.



You may specify as many or as few selection criteria as you wish. When you have completed your selection, click on the *Select Jobs* button above the HOSTS panel to have `xpbs` refresh the display with the jobs that match your selection criteria. The selected criteria will remain in effect until you change them again. If you exit `xpbs`, you will be prompted if you wish to save your configuration information; this includes the job selection criteria.

6.5 Using `xpbs` TrackJob Feature

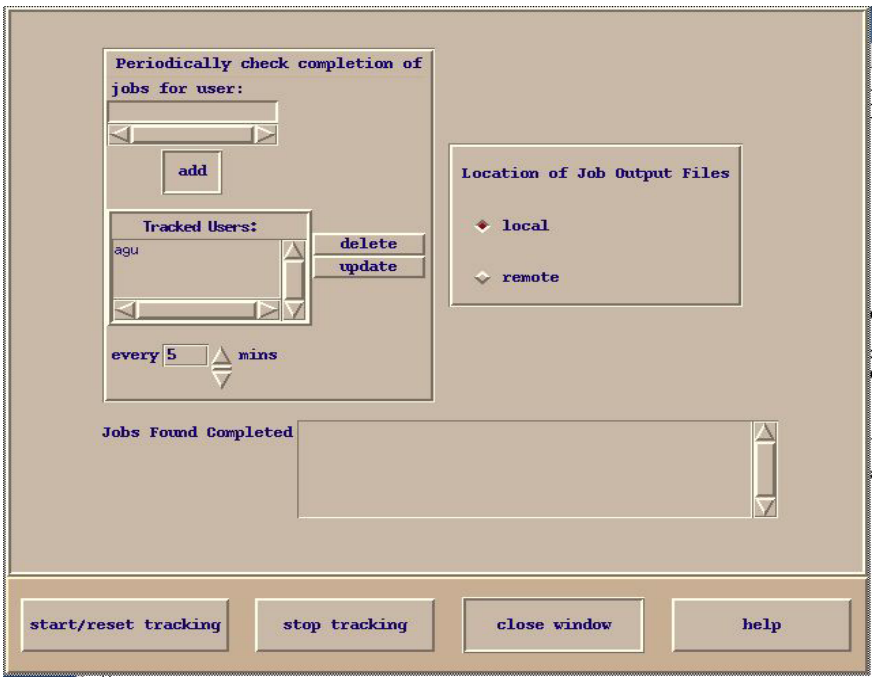
The `xpbs` command includes a feature that allows you to track the progress of your jobs. When you enable the *Track Job* feature, `xpbs` will monitor your jobs, looking for the output files that signal completion of the job. The *Track Job* button will flash red on the `xpbs` main display, and if you then click it, `xpbs` will display a list of all completed jobs (that you

were previously tracking). Selecting one of those jobs will launch a window containing the standard output and standard error files associated with the job.

IMPORTANT:

The Track Job feature is not currently available on Windows.

To enable `xpbs` job tracking, click on the *Track Job* button at the top center of the main `xpbs` display. Doing so will bring up the Track Job dialog box shown below.



From this window you can name the users whose jobs you wish to monitor. You also need to specify where you expect the output files to be: either local or remote (e.g. will the files be retained on the Server host, or did you request them to be delivered to another host?). Next, click the *start/reset tracking button* and then the *close window* button. Note that you can disable job tracking at any time by clicking the *Track Job* button on the main `xpbs` display, and then clicking the *stop tracking* button.

Chapter 7

Advanced PBS Features

This chapter covers the less commonly used commands and more complex topics which will add substantial functionality to your use of PBS. The reader is advised to read chapters 5 - 7 of this manual first.

7.1 UNIX Job Exit Status

On UNIX systems, the exit status of a job is normally the exit status of the shell executing the job script. If a user is using `csh` and has a `.logout` file in the home directory, the exit status of `csh` becomes the exit status of the last command in `.logout`. This may impact the use of job dependencies which depend on the job's exit status. To preserve the job's exit status, the user may either remove `.logout` or edit it as shown in this example:

```
set EXITVAL = $status
[ .logout's original content ]
```

Doing so will ensure that the exit status of the job persists across the invocation of the `.logout` file.

For a description of what job exit codes mean, see section 12.8, "Job Exit Codes", on page 857 of the PBS Professional Administrator's Guide.

The exit status of a job array is determined by the status of each of the completed subjobs. It is only available when all valid subjobs have completed. The individual exit status of a completed subjob is passed to the epilogue, and is available in the 'E' accounting log record of that subjob. See "Job Array Exit Status" on page 227.

7.2 Changing UNIX Job umask

The "`-W umask=`*nnn*" option to `qsub` allows you to specify, on UNIX systems, what `umask` PBS should use when creating and/or copying your `stdout` and `stderr` files, and any other files you direct PBS to transfer on your behalf.

IMPORTANT:

This feature does not apply to Windows.

The following example illustrates how to set your `umask` to 022 (i.e. to have files created with write permission for owner only: `-rw-r--r--`).

```
qsub -W umask=022 my_job
#PBS -W umask=022
```

7.3 Requesting qsub Wait for Job Completion

The "`-W block=true`" option to `qsub` allows you to specify that you want `qsub` to wait for the job to complete (i.e. "block") and report the exit value of the job. If job submission fails, no special processing will take place. If the job is successfully submitted, `qsub` will block until the job terminates or an error occurs.

If qsub receives one of the signals: SIGHUP, SIGINT, or SIGTERM, it will print a message and then exit with the exit status 2. If the job is deleted before running to completion, or an internal PBS error occurs, an error message describing the situation will be printed to this error stream and qsub will exit with an exit status of 3. Signals SIGQUIT and SIGKILL are not trapped and thus will immediately terminate the qsub process, leaving the associated job either running or queued. If the job runs to completion, qsub will exit with the exit status of the job. (See also section 7.1 "UNIX Job Exit Status" on page 159 for further discussion of the job exit status.)

For job arrays, blocking qsub waits until the entire job array is complete, then returns the exit status of the job array.

7.4 Specifying Job Dependencies

PBS allows you to specify dependencies between two or more jobs. Dependencies are useful for a variety of tasks, such as:

1. Specifying the order in which jobs in a set should execute
2. Requesting a job run only if an error occurs in another job
3. Holding jobs until a particular job starts or completes execution

The "-W depend=dependency_list" option to qsub defines the dependency between multiple jobs. The *dependency_list* has the format:

type:arg_list[,type:arg_list ...]

where except for the on type, the arg_list is one or more PBS job IDs in the form:

jobid[:jobid ...]

There are several types:

after:arg_list
: This job may be scheduled for execution at any point after all jobs in arg_list have started execution.

afterok:arg_list
: This job may be scheduled for execution only after all jobs in arg_list have terminated with no errors. See "Warning about exit status with csh" in EXIT STATUS.

afternotok:arg_list
: This job may be scheduled for execution only after all jobs in arg_list have terminated with errors. See "Warning about exit status with csh" in EXIT STATUS.

afterany:arg_list
: This job may be scheduled for execution after all jobs in arg_list have finished execution, with or without errors.

before:arg_list
: Jobs in arg_list may begin execution once this job has begun execution.

beforeok:arg_list
: Jobs in arg_list may begin execution once this job terminates without errors. See "Warning about exit status with csh" in EXIT STATUS.

beforenotok:arg_list
: If this job terminates execution with errors, the jobs in arg_list may begin. See "Warning about exit status with csh" in EXIT STATUS.

beforeany:arg_list
: Jobs in arg_list may begin execution once this job terminates execution, with or without errors.

on:count
: This job may be scheduled for execution after `count` dependencies on other jobs have been satisfied. This type is used in conjunction with one of the `before` types listed. `count` is an integer greater than 0.

Job IDs in the arg_list of `before` types must have been submitted with a type of `on`.

To use the `before` types, the user must have the authority to alter the jobs in `arg_list`. Otherwise, the dependency is rejected and the new job aborted.

Error processing of the existence, state, or condition of the job on which the newly submitted job is a deferred service, i.e. the check is performed after the job is queued. If an error is detected, the new job will be deleted by the server. Mail will be sent to the job submitter stating the error.

Suppose you have three jobs (job1, job2, and job3) and you want job3 to start *after* job1 and job2 have *ended*. The first example below illustrates the options you would use on the `qsub` command line to specify these job dependencies.

```
qsub job1
16394.jupiter
qsub job2
16395.jupiter
qsub -W depend=afterany:16394:16395 job3
16396.jupiter
```

As another example, suppose instead you want job2 to start *only if* job1 ends with no errors (i.e. it exits with a no error status):

```
qsub job1
16397.jupiter
qsub -W depend=afterok:16397 job2
16396.jupiter
```

Similarly, you can use `before` dependencies, as the following example exhibits. Note that unlike `after` dependencies, `before` dependencies require the use of the `on` dependency.

```
qsub -W depend=on:2 job1
16397.jupiter
qsub -W depend=beforeany:16397 job2
16398.jupiter
qsub -W depend=beforeany:16397 job3
16399.jupiter
```

You can use `xpbs` to specify job dependencies as well. On the *Submit Job* window, in the other options section (far left, center of window) click on one of the three dependency buttons: “after depend”, “before depend”, or “concurrency”. These will launch a “Dependency” window in which you will be able to set up the dependencies you wish.

7.4.1 Job Array Dependencies

Job dependencies are supported:

- Between jobs and jobs
- Between job arrays and job arrays
- Between job arrays and jobs
- Between jobs and job arrays

Note: Job dependencies are not supported for subjobs or ranges of subjobs.

7.4.2 Caveats and Advice for Job Dependencies

7.4.2.1 Warning About Exit Status

The exit status of the job can affect how the job’s dependencies work. If your `.logout` runs a command that sets the job’s exit status, you may need to change it. See section 7.1 “UNIX Job Exit Status” on page 159.

7.4.2.2 Warning About Job History

Enabling job history changes the behavior of dependent jobs. If a job j1 depends on a finished job j2 for which PBS is maintaining history than j1 will go into the held state. If job j1 depends on a finished job j3 that has been purged from the historical records than j1 will be rejected just as in previous versions of PBS where the job was no longer in the system.

7.5 Delivery of Output Files

To transfer output files or to transfer staged-in or staged-out files to/from a remote destination, PBS uses either `rcp` or `scp` depending on the configuration options. The version of `rcp` used by PBS always exits with a non-zero exit status for any error. Thus MOM knows if the file was delivered or not. The secure copy program, `scp`, is also based on this version of `rcp` and exits with the proper exit status.

If using `rcp`, the copy of output or staged files can fail for (at least) two reasons.

- The user lacks authorization to access the specified system. (See discussion in "User's PBS Environment" on page 13.)
- Under UNIX, if the user's `.cshrc` outputs any characters to standard output, e.g. contains an echo command, the copy will fail.

If using *Secure Copy (*`scp`), then PBS will first try to deliver output or stagein/out files using `scp`. If `scp` fails, PBS will try again using `rcp` (assuming that `scp` might not exist on the remote host). If `rcp` also fails, the above cycle will be repeated after a delay, in case the problem is caused by a temporary network problem. All failures are logged in MOM's log, and an email containing the errors is sent to the job owner.

For delivery of output files on the local host, PBS uses the `cp` command (UNIX) or the `xcopy` command (Windows XP) or the `robocopy` command (Windows Vista). Local and remote delivery of output may fail for the following additional reasons:

- A directory in the specified destination path does not exist.
- A directory in the specified destination path is not searchable by the user.
- The target directory is not writable by the user.

7.6 Input/Output File Staging

File staging is a way to specify which files should be copied onto the execution host before the job starts, and which should be copied off the execution host when it finishes.

7.6.1 Staging and Execution Directory: User's Home vs. Job-specific

The job's staging and execution directory is the directory to which files are copied before the job runs, and from which output files are copied after the job has finished. This directory is either your home directory or a job-specific directory created by PBS just for this job. If you use job-specific staging and execution directories, you don't need to have a home directory on each execution host, as long as those hosts are configured properly. In addition, each job gets its own staging and execution directory, so you can more easily avoid filename collisions.

This table lists the differences between using your home directory for staging and execution and using a job-specific staging and execution directory created by PBS.

Table 7-1: Differences Between User's Home and Job-specific Directory for Staging and Execution

Question Regarding Action, Requirement, or Setting	User's Home Directory	Job-specific Directory
Does PBS create a job-specific staging and execution directory?	No	Yes
User's home directory must exist on execution host(s)?	Yes	No
Standard out and standard error automatically deleted when qsub -k option is used?	No	Yes

Table 7-1: Differences Between User's Home and Job-specific Directory for Staging and Execution

Question Regarding Action, Requirement, or Setting	User's Home Directory	Job-specific Directory
When are staged-out files are deleted?	Successfully staged-out files are deleted; others go to "undelivered"	Only after all are successfully staged out
Staging and execution directory deleted after job finishes?	No	Yes
How is job's sandbox attribute set?	`HOME` or not set	PRIVATE

7.6.2 Using Job-specific Staging and Execution Directories

7.6.2.1 Setting the Job's Staging and Execution Directory

The job's `sandbox` attribute controls whether PBS creates a unique job-specific staging and execution directory for this job. If the job's `sandbox` attribute is set to `PRIVATE`, PBS creates a unique staging and execution directory for the job. If `sandbox` is unset, or is set to HOME, PBS uses the user's home directory as the job's staging and execution directory. By default, the `sandbox` attribute is not set.

The user can set the `sandbox` attribute via `qsub`, or through a PBS directive. For example:

```
qsub -Wsandbox=PRIVATE
```

The job's `sandbox` attribute cannot be altered while the job is executing.

Table 7-2: Effect of Job's `sandbox` Attribute on Location of Staging and Execution Directory

Job's `sandbox` attribute	Effect
not set	Job's staging and execution directory is the user's home directory
HOME	Job's staging and execution directory is the user's home directory
PRIVATE	Job's staging and execution directory is a job-specific directory created by PBS. If the qsub -k option is used, output and error files are retained on the primary execution host in the staging and execution directory. This directory is removed, along with all of its contents, when the job finishes.

7.6.2.2 The Job's `jobdir` Attribute and the PBS_JOBDIR Environment Variable

The job's `jobdir` attribute is a read-only attribute, set to the pathname of the job's staging and execution directory on the primary host. The user can view this attribute by using `qstat -f`, only while the job is executing. The value of `jobdir` is not retained if a job is rerun; it is undefined whether `jobdir` is visible or not when the job is not executing.

The environment variable `PBS_JOBDIR` is set to the pathname of the staging and execution directory on the primary execution host. `PBS_JOBDIR` is added to the job script process, any job tasks, and the prologue and epilogue.

7.6.3 Attributes and Environment Variables Affecting Staging

The following attributes and environment variables affect staging and execution.

Table 7-3: Attributes and Environment Variables Affecting Staging

Job's Attribute or Environment Variable	Effect
`sandbox` attribute	Determines whether PBS uses user's home directory or creates job-specific directory for staging and execution. User-settable per job via `qsub -W` or through a PBS directive.
`stagein` attribute	Sets list of files or directories to be staged in. User-settable per job via `qsub -W` or through a PBS directive.
`stageout` attribute	Sets list of files or directories to be staged out. User-settable per job via `qsub -W` or through a PBS directive.
`Keep_Files` attribute	Determines whether output and/or error files remain on execution host. User-settable per job via `qsub -k` or through a PBS directive. If the `Keep_Files` attribute is set to `o` and/or `e` (output and/or error files remain in the staging and execution directory) and the job's `sandbox` attribute is set to `PRIVATE`, standard out and/or error files are removed, when the staging directory is removed at job end along with its contents.
`jobdir` attribute	Set to pathname of staging and execution directory on primary execution host. Read-only; viewable via `qstat -f`.

Table 7-3: Attributes and Environment Variables Affecting Staging

Job's Attribute or Environment Variable	Effect
`PBS_JOBDIR` environment variable	Set to pathname of staging and execution directory on primary execution host. Added to environments of job script process, job tasks, and prologue and epilogue.
`TMPDIR` environment variable	Location of job-specific scratch directory.

7.6.4 Specifying Files To Be Staged In or Staged Out

You can specify files to be staged in before the job runs and staged out after the job runs by using `-W stagein=file_list` and `-W stageout=file_list`. You can use these as options to qsub, or as directives in the job script.

The *file_list* takes the form:

execution_path@hostname:storage_path[,...]

for both stagein and stageout.

The name *execution_path* is the name of the file in the job's staging and execution directory (on the execution host). The `execution_path` can be relative to the job's staging and execution directory, or it can be an absolute path.

The '@' character separates the execution specification from the storage specification.

The name *storage_path* is the file name on the host specified by *hostname*. For stagein, this is the location where the input files come from. For stageout, this is where the output files end up when the job is done. You must specify a hostname. The name can be absolute, or it can be relative to the user's home directory on the machine named *hostname*.

IMPORTANT:

It is advisable to use an absolute pathname for the storage_path. Remember that the path to your home directory may be different on each machine, and that when using `sandbox = PRIVATE,` you may or may not have a home directory on all execution machines.

For stagein, the direction of travel is **from** storage_path **to** execution_path.

For stageout, the direction of travel is **from** execution_path **to** storage_path.

The following example shows how to use a directive to stagein a file named `grid.dat` located in the directory `/u/user1` on the host called `serverA`. The staged-in file is copied to the staging and execution directory and given the name `dat1`. Since `execution_path` is evaluated relative to the staging and execution directory, it is not necessary to specify a full pathname for `dat1`. Always use a relative pathname for `execution_path` when the job's staging and execution directory is created by PBS.

```
#PBS -W stagein=dat1@serverA:/u/user1/grid.dat ...
```

To use the `qsub` option to stage in the file residing on myhost, in `/Users/myhome/mydata/data1`, calling it `input_data1` in the staging and execution directory:

```
qsub -W stagein=input_data1@myhost:/Users/myhome/
   mydata/data1
```

To stage more than one file or directory, use a comma-separated list of paths, and enclose the list in double quotes. For example, to stage two files data1 and data2 in:

```
qsub -W stagein="input1@hostA:/myhome/data1,
   \input2@hostA:/myhome/data1"
```

- Under Windows, special characters such as spaces, backslashes (\), colons (:), and drive letter specifications are valid pathnames. For example, the following will stagein the grid.dat file on drive D at hostB to a local file ("dat1") on drive C.:

```
qsub -W stagein="dat1@hostB:D\Documents and Set-
   tings\grid.dat"
```

7.6.4.1 Copying Directories Into and Out Of the Staging and Execution Directory

You can stage directories into and out of the staging and execution directory the same way you stage files. The *storage_path* and *execution_path* for both stagein and stageout can be a directory. If you stagein or stageout a directory, PBS copies that directory along with all of its files and subdirectories. At the end of the job, the directory, including all files and subdirectories, is deleted. This can create a problem if multiple jobs are using the same directory.

7.6.4.2 Wildcards In File Staging

You can use wildcards when staging files and directories, according to the following rules.

- The asterisk "*" matches one or more characters.
- The question mark "?" matches a single character.
- All other characters match only themselves.
- Wildcards inside of quote marks are expanded.
- Wildcards cannot be used to match UNIX files that begin with period "." or Windows files that have the "SYSTEM" or "HIDDEN" attributes.
- When using the qsub command line on UNIX, you must prevent the shell from expanding wildcards. For some shells, you can enclose the pathnames in double quotes. For some shells, you can use a backspace before the wildcard.
- Wildcards can only be used in the source side of a staging specification. This means they can be used in the storage_path specification for stagein, and in the execution_path specification for stageout.
- When staging using wildcards, the destination must be a directory. If the destination is not a directory, the result is undefined. So for example, when staging out all .out files, you must specify a directory for storage_path.
- Wildcards can only be used in the final path component, i.e. the basename.

- When wildcards are used during stagein, PBS will not automatically delete staged files at job end. Note that if PBS created the staging and execution directory, that directory and all its contents are deleted at job end.

7.6.4.3 Examples of File Staging

Example 1: Stage out all files from the execution directory to a specific directory:

UNIX

```
-W stageout=*@myworkstation:/user/project1/case1
```

Windows

```
-W stageout=*@mypc:E:\project1\case1
```

Example 2: Stage out specific types of result files and disregard the scratch and other temporary files after the job terminates. The result files that are interesting for this example end in '.dat':

UNIX

```
-W stageout=*.dat@myworkstation:project3/data
```

Windows

```
-W stageout=*.dat@mypc:C:\project\data
```

Example 3: Stage in all files from an application data directory to a subdirectory:

UNIX

```
-W stagein=jobarea@myworkstation:crashtest1/*
```

Windows

```
-W stagein=jobarea@mypc:E:\crashtest1\*
```

Example 4: Stage in data from files and directories matching “wing*”:

UNIX

```
-W stagein=.@myworkstation:848/wing*
```

Windows

```
-W stagein=.@mypc:E:\flowcalc\wing*
```

Example 5: Stage in .bat and .dat files to jobarea:

UNIX:

```
-W stagein=jobarea@myworkstation:/users/me/crash1.?at
```

Windows:

```
-W stagein=jobarea@myworkstation:C:\me\crash1.?at
```

7.6.4.4 Caveats

When using a job-specific staging and execution directory, do not use an absolute path in `execution_path`.

7.6.4.5 Output Filenames

The name of the job defaults to the script name, if no name is given via `qsub -N`, via a PBS directive, or via stdin. For example, if the sequence number is 1234,

```
#PBS -N fixgamma
```

gives stdout the name fixgamma.o1234 and stderr the name fixgamma.e1234.

For information on submitting jobs, see section 3.4 “Submitting a PBS Job” on page 33.

7.6.5 Example of Using Job-specific Staging and Execution Directories

In this example, you want the file “jay.fem” to be delivered to the job-specific staging and execution directory given in PBS_JOBDIR, by being copied from the host “submithost”. The job script is executed in PBS_JOBDIR and “jay.out” is staged out from PBS_JOBDIR to your home directory on the submittal host (i.e., “hostname”):

```
qsub -Wsandbox=PRIVATE -Wstagein=jay.fem@submit-
   host:jay.fem -Wstageout=jay.out@submithost:jay.out
```

7.6.6 Summary of the Job's Lifecycle

This is a summary of the steps performed by PBS. The steps are not necessarily performed in this order.

- On each execution host, if specified, PBS creates a job-specific staging and execution directory.
- PBS sets PBS_JOBDIR and the job's jobdir attribute to the path of the job's staging and execution directory.
- On each execution host allocated to the job, PBS creates a job-specific temporary directory.
- PBS sets the TMPDIR environment variable to the pathname of the temporary directory.
- If any errors occur during directory creation or the setting of variables, the job is requeued.
- PBS stages in any files or directories.
- The prologue is run on the primary execution host, with its current working directory set to PBS_HOME/mom_priv, and with PBS_JOBDIR and TMPDIR set in its environment.
- The job is run as the user on the primary execution host.
- The job's associated tasks are run as the user on the execution host(s).
- The epilogue is run on the primary execution host, with its current working directory set to the path of the job's staging and execution directory, and with PBS_JOBDIR and TMPDIR set in its environment.
- PBS stages out any files or directories.
- PBS removes any staged files or directories.
- PBS removes any job-specific staging and execution directories and their contents, and all TMPDIRs and their contents.
- PBS writes the final job accounting record and purges any job information from the Server's database.

7.6.7 Detailed Description of Job's Lifecycle

7.6.7.1 Creation of TMPDIR

For each host allocated to the job, PBS creates a job-specific temporary scratch directory for the job. If the temporary scratch directory cannot be created, the job is aborted.

7.6.7.2 Choice of Staging and Execution Directories

If the job's `sandbox` attribute is set to `PRIVATE`, PBS creates job-specific staging and execution directories for the job. If the job's `sandbox` attribute is set to `HOME`, or is unset, PBS uses the user's home directory for staging and execution.

7.6.7.2.1 Job-specific Staging and Execution Directories

If the staging and execution directory cannot be created the job is aborted. If PBS fails to create a staging and execution directory, see the system administrator.

You should not depend on any particular naming scheme for the new directories that PBS creates for staging and execution.

7.6.7.2.2 User's Home Directory as Staging and Execution Directory

The user must have a home directory on each execution host. The absence of the user's home directory is an error and causes the job to be aborted.

7.6.7.3 Setting Environment Variables and Attributes

PBS sets `PBS_JOBDIR` and the job's `jobdir` attribute to the pathname of the staging and execution directory. The TMPDIR environment variable is set to the pathname of the job-specific temporary scratch directory.

7.6.7.4 Staging Files Into Staging and Execution Directories

PBS evaluates `execution_path` and `storage_path` relative to the staging and execution directory given in PBS_JOBDIR, whether this directory is the user's home directory or a job-specific directory created by PBS. PBS copies the specified files and/or directories to the job's staging and execution directory.

7.6.7.5 Running the Prologue

The MOM's prologue is run on the primary host as root, with the current working directory set to `PBS_HOME/mom_priv`, and with `PBS_JOBDIR` and `TMPDIR` set in its environment.

7.6.7.6 Job Execution

PBS runs the job script on the primary host as the user. PBS also runs any tasks created by the job as the user. The job script and tasks are executed with their current working directory set to the job's staging and execution directory, and with `PBS_JOBDIR` and `TMPDIR` set in their environment.

7.6.7.7 Standard Out and Standard Error

The job's stdout and stderr files are created directly in the job's staging and execution directory on the primary execution host.

7.6.7.7.1 Job-specific Staging and Execution Directories

If the `qsub -k` option is used, the stdout and stderr files will **not** be automatically copied out of the staging and execution directory at job end - they will be deleted when the directory is automatically removed.

7.6.7.7.2 User's Home Directory as Staging and Execution Directory

If the `-k` option to `qsub` is used, standard out and/or standard error files are retained on the primary execution host instead of being returned to the submission host, and are not deleted after job end.

7.6.7.8 Running the Epilogue

PBS runs the epilogue on the primary host as root. The epilogue is executed with its current working directory set to the job's staging and execution directory, and with `PBS_JOBDIR` and `TMPDIR` set in its environment.

7.6.7.9 Staging Files Out and Removing Execution Directory

When PBS stages files out, it evaluates `execution_path` and `storage_path` relative to `PBS_JOBDIR`. Files that cannot be staged out are saved in PBS_HOME/undelivered. See section 14.5.6, "Non-delivery of Output" on page 912 in the PBS Professional Administrator's Guide.

7.6.7.9.1 Job-specific Staging and Execution Directories

If PBS created job-specific staging and execution directories for the job, it cleans up at the end of the job. The staging and execution directory and all of its contents are removed, on all execution hosts.

7.6.7.10 Removing TMPDIRs

PBS removes all TMPDIRs, along with their contents.

7.6.8 Staging with Job Arrays

File staging is supported for job arrays. See "File Staging" on page 210.

7.6.9 Using xpbs for File Staging

Using `xpbs` to set up file staging directives may be easier than using the command line. On the *Submit Job* window, in the miscellany options section (far left, center of window) click on the *file staging* button. This will launch the *File Staging* dialog box (shown below) in which you will be able to set up the file staging you desire.

The *File Selection Box* will be initialized with your current working directory. If you wish to select a different directory, double-click on its name, and `xpbs` will list the contents of the new directory in the *File Selection Box*. When the correct directory is displayed, simply click on the name of the file you wish to stage (in or out). Its name will be written in the *File Selected* area.

Next, click either of the *Add file selected...* buttons to add the named file to the stagein or stageout list. Doing so will write the file name into the corresponding area on the lower half of the *File Staging* window. Now you need to provide location information. For stagein, type in the path and filename where you want the named file placed. For stageout, specify the hostname and pathname where you want the named file delivered. You may repeat this process for as many files as you need to stage.

When you are done selecting files, click the *OK* button.

7.6.10 Stagein and Stageout Failure

When stagein fails, the job is placed in a 30-minute wait to allow the user time to fix the problem. Typically this is a missing file or a network outage. Email is sent to the job owner when the problem is detected. Once the problem has been resolved, the job owner or the Operator may remove the wait by resetting the time after which the job is eligible to be run via the `-a` option to `qalter`. The server will update the job's comment with information about why the job was put in the wait state. When the job is eligible to run, it may run on different vnodes.

When stageout encounters an error, there are three retries. PBS waits 1 second and tries

again, then waits 11 seconds and tries a third time, then finally waits another 21 seconds

and tries a fourth time. Email is sent to the job owner if all attempts fail. Files that cannot be staged out are saved in PBS_HOME/undelivered. See section 14.5.6, "Non-delivery of Output" on page 912 in the PBS Professional Administrator's Guide.

7.7 The pbsdsh Command

The **pbsdsh** command allows you to distribute and execute a task on each of the vnodes assigned to your job. (pbsdsh uses the PBS Task Manager API, see tm(3), to distribute the program on the allocated vnodes.)

IMPORTANT:

The pbsdsh command is not available under Windows.

Usage of the pbsdsh command is:

pbsdsh [-c N] [-o] [-s] [-v] -- program [program args]

pbsdsh [-n N] [-o] [-s] [-v] -- program [program args]

Note that the double dash must come after the options and before the program and arguments. The double dash is only required for Linux.

The available options are:

-c N

The program is spawned on the first *N* vnodes allocated. If the value of *N* is greater than the number of vnodes, it will "wrap" around, running multiple copies on the vnodes. This option is mutually exclusive with -n.

-n N

The program is spawned on a single vnode which is the *N*-th vnode allocated. This option is mutually exclusive with -c.

-o

The program will not wait for the tasks to finish.

-s

If this option is given, the program is run sequentially on each vnode, one after the other.

-v

Verbose output about error messages and task exit status is produced.

When run without the -c or the -n option, pbsdsh will spawn the program on all vnodes allocated to the PBS job. The execution take place concurrently--all copies of the task execute at (about) the same time.

The following example shows the pbsdsh command inside of a PBS batch job. The options indicate that the user wants pbsdsh to run the *myapp* program with one argument (*app-arg1*) on all four vnodes allocated to the job (i.e. the default behavior).

```
#!/bin/sh
#PBS -l select=4:ncpus=1
#PBS -l walltime=1:00:00


pbsdsh ./myapp app-arg1
```

The pbsdsh command runs one task for each line in the PBS_NODEFILE. Each MPI rank will get a single line in the PBS_NODEFILE, so if you are running multiple MPI ranks on the same host, you will still get multiple pbsdsh tasks on that host.

7.8 Advance and Standing Reservation of Resources

7.8.1 Definitions

Advance reservation
: A reservation for a set of resources for a specified time. The reservation is only available to a specific user or group of users.

Standing reservation
: An advance reservation which recurs at specified times. For example, the user can reserve 8 CPUs and 10GB every Wednesday and Thursday from 5pm to 8pm, for the next three months.

Occurrence of a standing reservation
: An instance of the standing reservation.

An occurrence of a standing reservation behaves like an advance reservation, with the following exceptions:

- while a job can be submitted to a specific advance reservation, it can only be submitted to the standing reservation as a whole, not to a specific occurrence. You can only specify *when* the job is eligible to run. See the `qsub(1B)` man page.
- when an advance reservation ends, it and all of its jobs, running or queued, are deleted, but when an occurrence ends, only its running jobs are deleted.

Each occurrence of a standing reservation has reserved resources which satisfy the resource request, but each occurrence may have its resources drawn from a different source. A query for the resources assigned to a standing reservation will return the resources assigned to the soonest occurrence, shown in the resv_nodes attribute reported by `pbs_rstat`.

Soonest occurrence of a standing reservation

The occurrence which is currently active, or if none is active, then it is the next occurrence.

Degraded reservation

An advance reservation for which one or more associated vnodes are unavailable.

A standing reservation for which one or more vnodes associated with any occurrence are unavailable.

7.8.2 Introduction to Creating and Using Reservations

The user creates both advance and standing reservations using the `pbs_rsub` command. PBS either confirms that the reservation can be made, or rejects the request. Once the reservation is confirmed, PBS creates a queue for the reservation's jobs. Jobs are then submitted to this queue.

When a reservation is confirmed, it means that the reservation will not conflict with currently running jobs, other confirmed reservations, or dedicated time, and that the requested resources are available for the reservation. A reservation request that fails these tests is rejected. All occurrences of a standing reservation must be acceptable in order for the standing reservation to be confirmed.

The `pbs_rsub` command returns a *reservation ID*, which is the reservation name. For an advance reservation, this reservation ID has the format:

R<unique integer>.<server name>

For a standing reservation, this reservation ID refers to the entire series, and has the format:

S<unique integer>.<server name>

The user specifies the resources for a reservation using the same syntax as for a job. Jobs in reservations are placed the same way non-reservation jobs are placed in placement sets.

The `xpbs` GUI cannot be used for creation, querying, or deletion of reservations.

The time for which a reservation is requested is in the time zone at the submission host.

7.8.3 Creating Advance Reservations

You create an advance reservation using the `pbs_rsub` command. PBS must be able to calculate the start and end times of the reservation, so you must specify two of the following three options:

D	Duration
E	End time
R	Start time

7.8.3.1 Examples of Creating Advance Reservations

The following example shows the creation of an advance reservation asking for 1 vnode, 30 minutes of wall-clock time, and a start time of 11:30. Since an end time is not specified, PBS will calculate the end time based on the reservation start time and duration.

```
pbs_rsub -R 1130 -D 00:30:00
```

PBS returns the reservation ID:

```
R226.south UNCONFIRMED
```

The following example shows an advance reservation for 2 CPUs from 8:00 p.m. to 10:00 p.m.:

```
pbs_rsub -R 2000.00 -E 2200.00 -l select=1:ncpus=2
```

PBS returns the reservation ID:

```
R332.south UNCONFIRMED
```

7.8.4 Creating Standing Reservations

You create standing reservations using the `pbs_rsub` command. You **must** specify a start and end date when creating a standing reservation. The recurring nature of the reservation is specified using the `-r` option to `pbs_rsub`. The `-r` option takes the `recurrence_rule` argument, which specifies the standing reservation's occurrences. The recurrence rule uses iCalendar syntax, and uses a subset of the parameters described in RFC 2445.

The recurrence rule can take two forms:

"FREQ= freq_spec; COUNT= count_spec; interval_spec"

In this form, you specify how often there will be occurrences, how many there will be, and which days and/or hours apply.

"FREQ= freq_spec; UNTIL= until_spec; interval_spec"

In this form, the user specifies how often there will be occurrences, when the occurrences will end, and which days and/or hours apply.

freq_spec

This is the frequency with which the reservation repeats. Valid values are WEEKLY | DAILY | HOURLY

When using a freq_spec of WEEKLY, you may use an interval_spec of BYDAY and/or BYHOUR. When using a freq_spec of DAILY, you may use an interval_spec of BYHOUR. When using a freq_spec of HOURLY, do not use an interval_spec.

count_spec

The exact number of occurrences. Number up to 4 digits in length. Format: integer.

interval_spec

Specifies the interval at which there will be occurrences. Can be one or both of BYDAY=<days> or BYHOUR=<hours>. Valid values are BYDAY = MO | TU | WE | TH | FR | SA | SU and BYHOUR = 0 | 1 | 2 | ... | 23. When using both, separate them with a semicolon. Separate days or hours with a comma.

For example, to specify that there will be recurrences on Tuesdays and Wednesdays, at 9 a.m. and 11 a.m., use BYDAY=TU,WE;BYHOUR=9,11

BYDAY should be used with FREQ=WEEKLY. BYHOUR should be used with FREQ=DAILY or FREQ=WEEKLY.

until_spec

Occurrences will start up to but not after this date and time. This means that if occurrences last for an hour, and normally start at 9 a.m., then a time of 9:05 a.m on the day specified in the until_spec means that an occurrence will start on that day.

Format: YYYYMMDD[THHMMSS]

Note that the year-month-day section is separated from the hour-minute-second section by a capital T.

Default: 3 years from time of reservation creation.

7.8.4.1 Setting Reservation Start Time and Duration

In a standing reservation, the arguments to the -R and -E options to pbs_rsub can provide more information than they do in an advance reservation. In an advance reservation, they provide the start and end time of the reservation. In a standing reservation, they can provide the start and end time, but they can also be used to compute the duration and the offset from the interval start.

The difference between the values of the arguments for -R and -E is the duration of the reservation. For example, if you specify

```
-R 0930 -E 1145
```

the duration of your reservation will be two hours and fifteen minutes. If you specify

```
-R 150800 -E 170830
```

the duration of your reservation will be two days plus 30 minutes.

The interval_spec can be used to specify the day or the hour at which the interval starts. If you specify

```
-R 0915 -E 0945 ... BYHOUR=9,10
```

the duration is 30 minutes, and the offset is 15 minutes from the start of the interval. The interval start is at 9 and again at 10. Your reservation will run from 9:15 to 9:45, and again at 10:15 and 10:45. Similarly, if you specify

```
-R 0800 -E -1000 ... BYDAY=WE,TH
```

the duration is two hours and the offset is 8 hours from the start of the interval. Your reservation will run Wednesday from 8 to 10, and again on Thursday from 8 to 10.

Elements specified in the recurrence rule override those specified in the arguments to the -R and -E options. Therefore if you specify

```
-R 0730 -E 0830 ... BYHOUR=9
```

the duration is one hour, but the hour element (9:00) in the recurrence rule has overridden the hour element specified in the argument to -R (7:00). The offset is still 30 minutes after the interval start. Your reservation will run from 9:30 to 10:30. Similarly, if the 16th is a Monday, and you specify

```
-R 160800 -E 170900 ... BYDAY=TU;BYHOUR=11
```

the duration 25 hours, but both the day and the hour elements have been overridden. Your reservation will run on Tuesday at 11, for 25 hours, ending Wednesday at 12. However, if you specify

```
-R 160810 -E 170910 ... BYDAY=TU;BYHOUR=11
```

the duration is 25 hours, and the offset from the interval start is 10 minutes. Your reservation will run on Tuesday at 11:10, for 25 hours, ending Wednesday at 12:10. The minutes in the offset weren't overridden by anything in the recurrence rule.

The values specified for the arguments to the -R and -E options can be used to set the start and end times in a standing reservation, just as they are in an advance reservation. To do this, don't override their elements inside the recurrence rule. If you specify

```
-R 0930 -E 1030 ... BYDAY=MO,TU
```

you haven't overridden the hour or minute elements. Your reservation will run Monday and Tuesday, from 9:30 to 10:30.

7.8.4.2 Requirements for Creating Standing Reservations

- The user must specify a start and end date. See the -R and -E options to the pbs_rsub command in section 7.8.5 "The pbs_rsub Command" on page 188.
- The user must set the submission host's PBS_TZID environment variable. The format for PBS_TZID is a timezone location. Example: America/Los_Angeles, America/Detroit, Europe/Berlin, Asia/Calcutta. See section 7.8.9.1 "Setting the Submission Host's Time Zone" on page 196.
- The recurrence rule must be one unbroken line. See the -r option to pbs_rsub in section 7.8.5 "The pbs_rsub Command" on page 188.
- The recurrence rule must be enclosed in double quotes.
- Vnodes that have been configured to accept jobs only from a specific queue (vnode-queue restrictions) cannot be used for advance or standing reservations. See your PBS administrator to determine whether some vnodes have been configured to accept jobs only from specific queues.

7.8.4.3 Examples of Creating Standing Reservations

For a reservation that runs every day from 8am to 10am, for a total of 10 occurrences:

```
pbs_rsub -R 0800 -E 1000 - r"FREQ=DAILY;COUNT=10"
```

Every weekday from 6am to 6pm until December 10, 2008:

```
pbs_rsub -R 0600 -E 1800 -r "FREQ=WEEKLY;
   BYDAY=MO,TU,WE,TH,FR; UNTIL=20081210"
```

Every week from 3pm to 5pm on Monday, Wednesday, and Friday, for 9 occurrences, i.e., for three weeks:

```
pbs_rsub -R 1500 -E 1700 -r
   "FREQ=WEEKLY;BYDAY=MO,WE,FR; COUNT=9"
```

7.8.5 The pbs_rsub Command

The pbs_rsub command returns a reservation ID string, and the current status of the reservation.

For the options to the pbs_rsub command, see “pbs_rsub” on page 95 of the PBS Professional Reference Guide.

7.8.5.1 Getting Confirmation of a Reservation

By default the pbs_rsub command does not immediately notify you whether the reservation is confirmed or denied. Instead you receive email with this information. You can specify that the pbs_rsub command should wait for confirmation by using the -I *<block_time>* option. The pbs_rsub command will wait up to *<block_time>* seconds for the reservation to be confirmed or denied and then notify you of the outcome. If *block_time* is negative and the reservation is not confirmed in that time, the reservation is automatically deleted.

To find out whether the reservation has been confirmed, use the pbs_rstat command. It will display the state of the reservation. CO and RESV_CONFIRMED indicate that it is confirmed. If the reservation does not appear in the output from pbs_rstat, that means that the reservation was denied.

To ensure that you receive mail about your reservations, set the reservation's `Mail_Users` attribute via the `-M` <*email address*> option to `pbs_rsub`. By default, you will get email when the reservation is terminated or confirmed. If you want to receive email about events other than those, set the reservation's `Mail_Points` attribute via the `-m` <*mail events*> option. For more information, see the `pbs_rsub(1B)` and `pbs_resv_attributes(7B)` man pages.

7.8.6 Viewing the Status of a Reservation

The following table shows the list of possible states for a reservation. The states that you will usually see are `CO, UN, BD`, and `RN`, although a reservation usually remains unconfirmed for too short a time to see that state. See "Reservation States" on page 472 of the PBS Professional Reference Guide.

To view the status of a reservation, use the `pbs_rstat` command. It will display the status of all reservations at the PBS server. For a standing reservation, the `pbs_rstat` command will display the status of the soonest occurrence. Duration is shown in seconds. The `pbs_rstat` command will not display a custom resource which has been created to be invisible. See section 3.5.15 "Resource Permissions" on page 49. This command has three options:

Table 7-4: Options to `pbs_rstat` Command

Option	Meaning	Description
B	Brief	Lists only the names of the reservations
S	Short	Lists in table format the name, queue name, owner, state, and start, duration and end times of each reservation
F	Full	Lists the name and all non-default-value attributes for each reservation.
<none>	Default	Default is S option

The full listing for a standing reservation is identical to the listing for an advance reservation, with the following additions:

- A line that specifies the recurrence rule:

  ```
  reserve_rrule = FREQ=WEEKLY;BYDAY=MO;COUNT=5
  ```

- An entry for the vnodes reserved for the soonest occurrence of the standing reservation. This entry also appears for an advance reservation, but will be different for each occurrence:

  ```
  resv_nodes=(vnode_name:...)
  ```

- A line that specifies the total number of occurrences of the standing reservation:

  ```
  reserve_count = 5
  ```

- The index of the soonest occurrence:

  ```
  reserve_index = 1
  ```

- The timezone at the site of submission of the reservation is appended to the reservation Variable list. For example, in California:

  ```
  Variable_List=<other variables>PBS_TZID=America/
     Los_Angeles
  ```

To get the status of a reservation at a server other than the default server, set the `PBS_SERVER` environment variable to the name of the server you wish to query, then use the `pbs_rstat` command. Your PBS commands will treat the new server as the default server, so you may wish to unset this environment variable when you are finished.

You can also get information about the reservation's queue by using the `qstat` command. See "qstat" on page 228 of the PBS Professional Reference Guide and the `qstat(1B)` man page.

7.8.6.1 Examples of Viewing Reservation Status Using `pbs_rstat`

In our example, we have one advance reservation and one standing reservation. The advance reservation is for today, for two hours, starting at noon. The standing reservation is for every Thursday, for one hour, starting at 3:00 p.m. Today is Monday, April 28th, and the time is 1:00, so the advance reservation is running, and the soonest occurrence of the standing reservation is Thursday, May 1, at 3:00 p.m.

Example brief output:

```
pbs_rstat -B
Name: R302.south
Name: S304.south
```

Example short output:

```
pbs_rstat -S

Name      Queue User State Start / Duration / End
---------------------------------------------------------
R302.south R302 user1 RN Today 12:00 / 7200/  Today 14:00
S304.south S304 user1 CO May 1 2008 15:00/3600/May 1 2008 16:00
```

Example full output:

```
pbs_rstat -F
```

```
Name: R302.south
Reserve_Name = NULL
Reserve_Owner = user1@south.mydomain.com
reserve_state = RESV_RUNNING
reserve_substate = 5
reserve_start = Mon Apr 28 12:00:00 2008
reserve_end = Mon Apr 28 14:00:00 2008
reserve_duration = 7200
queue = R302
Resource_List.ncpus = 2
Resource_List.nodect = 1
Resource_List.walltime = 02:00:00
Resource_List.select = 1:ncpus=2
Resource_List.place = free
resv_nodes = (south:ncpus=2)
Authorized_Users = user1@south.mydomain.com
server = south
ctime = Mon Apr 28 11:00:00 2008
Mail_Users = user1@mydomain.com
mtime = Mon Apr 28 11:00:00 2008
Variable_List =
   PBS_O_LOGNAME=user1,PBS_O_HOST=south.mydomain.com

Name: S304.south
Reserve_Name = NULL
Reserve_Owner = user1@south.mydomain.com
reserve_state = RESV_CONFIRMED
reserve_substate = 2
reserve_start = Thu May 1 15:00:00 2008
reserve_end = Thu May 1 16:00:00 2008
reserve_duration = 3600
```

```
queue = S304
Resource_List.ncpus = 2
Resource_List.nodect = 1
Resource_List.walltime = 01:00:00
Resource_List.select = 1:ncpus=2
Resource_List.place = free
resv_nodes = (south:ncpus=2)
reserve_rrule = FREQ=WEEKLY;BYDAY=MO;COUNT=5
reserve_count = 5
reserve_index = 2
Authorized_Users = user1@south.mydomain.com
server = south
ctime = Mon Apr 28 11:01:00 2008
Mail_Users = user1@mydomain.com
mtime = Mon Apr 28 11:01:00 2008
Variable_List =
   PBS_O_LOGNAME=user1,PBS_O_HOST=south.mydo-
   main.com,PBS_TZID=America/Los_Angeles
```

7.8.7 Deleting Reservations

You can delete an advance or standing reservation by using the `pbs_rdel` command. For a standing reservation, you can only delete the entire reservation, including all occurrences. When you delete a reservation, all of the jobs that have been submitted to the reservation are also deleted. A reservation can be deleted by its owner or by a PBS Operator or Manager. For example, to delete `S304.south`:

```
pbs_rdel S304.south
```

or

```
pbs_rdel S304
```

7.8.8 Submitting a Job to a Reservation

Jobs can be submitted to the queue associated with a reservation, or they can be moved from another queue into the reservation queue. You submit a job to a reservation by using the -q <*queue*> option to the `qsub` command to specify the reservation queue. For example, to submit a job to the soonest occurrence of a standing reservation named `S123.south`, submit to its queue `S123`:

```
qsub -q S123 <script>
```

You move a job into a reservation queue by using the `qmove` command. For more information, see the `qsub(1B)` and `qmove(1B)` man pages. For example, to `qmove` job `22.myhost` from `workq` to `S123`, the queue for the reservation named `S123.south`:

```
qmove S123 22.myhost
```

or

```
qmove S123 22
```

A job submitted to a standing reservation without a restriction on when it can run will be run, if possible, during the soonest occurrence. In order to submit a job to a specific occurrence, use the `-a` <*start time*> option to the `qsub` command, setting the start time to the time of the occurrence that you want. You can also use a `cron` job to submit a job at a specific time. See the `qsub(1B)` and `cron(8)` man pages.

7.8.8.1 Running Jobs in a Reservation

A confirmed reservation will accept jobs into its queue at any time. Jobs are only scheduled to run from the reservation once the reservation period arrives.

The jobs in a reservation are not allowed to use, in aggregate, more resources than the reservation requested. A reservation job is started only if its requested walltime will fit within the reservation period. So for example if the reservation runs from 10:00 to 11:00, and the job's walltime is 4 hours, the job will not be started.

When an advance reservation ends, any running or queued jobs in that reservation are deleted.

When an occurrence of a standing reservation ends, any running jobs in that reservation are killed. Any jobs still queued for that reservation are kept in the queued state. They are allowed to run in future occurrences. When the last occurrence of a standing reservation ends, all jobs remaining in the reservation are deleted, whether queued or running.

A job in a reservation cannot be preempted.

7.8.8.1.1 Reservation Fault Tolerance

If one or more vnodes allocated to an advance reservation or to the soonest occurrence of a standing reservation become unavailable, the reservation's state becomes *DG* or *RESV_DEGRADED*. A degraded reservation does not have all the reserved resources to run its jobs.

PBS attempts to reconfirm degraded reservations. This means that it looks for alternate available vnodes on which to run the reservation. The reservation's retry_time attribute lists the next time when PBS will try to reconfirm the reservation.

If PBS is able to reconfirm a degraded reservation, the reservation's state becomes *CO*, or *RESV_CONFIRMED*, and the reservation's resv_nodes attribute shows the new vnodes.

7.8.8.2 Access to Reservations

By default, the reservation accepts jobs only from the user who created the reservation, and accepts jobs submitted from any group or host. You can specify a list of users and groups whose jobs will and will not be accepted by the reservation by setting the reservation's `Authorized_Users` and `Authorized_Groups` attributes using the `-U` *auth_user_list* and `-G` *auth_group_list* options to `pbs_rsub`. You can specify the hosts from which jobs can and cannot be submitted by setting the reservation's `Authorized_Hosts` attribute using the `-H` *auth_host_list* option to `pbs_rsub`.

The administrator can also specify which users and groups can and cannot submit jobs to a reservation, and the list of hosts from which jobs can and cannot be submitted.

For more information, see the `pbs_rsub(1B)` and `pbs_resv_attributes(7B)` man pages.

7.8.8.3 Viewing Status of a Job Submitted to a Reservation

You can view the status of a job that has been submitted to a reservation or to an occurrence of a standing reservation by using the `qstat` command. See "qstat" on page 228 of the PBS Professional Reference Guide and the `qstat(1B)` man page.

For example, if a job named `MyJob` has been submitted to the soonest occurrence of the standing reservation named `S304.south,` it is listed under `S304`, the name of the queue:

```
qstat


Job id      Name       User          Time Use  S  Queue
----------  ---------  ------------  -------- --  -----
139.south   MyJob      user1                0  Q  S304
```

7.8.9 Reservation Caveats and Errors

7.8.9.1 Setting the Submission Host's Time Zone

The environment variable `PBS_TZID` must be set at the submission host. The time for which a reservation is requested is the time defined at the submission host. The format for `PBS_TZID` is a timezone location, rather than a timezone POSIX abbreviation. Examples of values for `PBS_TZID` are:

```
America/Los_Angeles
America/Detroit
Europe/Berlin
Asia/Calcutta
```

7.8.9.2 Reservation Errors

The following table describes the error messages that apply to reservations:

Table 7-5: Reservation Errors

Description of Error	Server Log Error Code	Error Message
Invalid syntax when specifying a standing reservation	15133	"pbs_rsub error: Undefined iCalendar syntax"
Recurrence rule has both a `COUNT` and an `UNTIL` parameter	15134	"pbs_rsub error: Undefined iCalendar syntax. COUNT or UNTIL is required"
Recurrence rule missing valid `COUNT` or `UNTIL` parameter	15134	"pbs_rsub error: Undefined iCalendar syntax. A valid COUNT or UNTIL is required"
Problem with the start and/or end time of the reservation, such as: Given start time is earlier than current date and time Missing start time or end time End time is earlier than start time	15086	"pbs_rsub: Bad time specification(s)"
Reservation duration exceeds 24 hours and the recurrence frequency, `FREQ`, is set to `DAILY`	15129	"pbs_rsub error: DAILY recurrence duration cannot exceed 24 hours"
Reservation duration exceeds 7 days and the frequency `FREQ` is set to `WEEKLY`	15128	"pbs_rsub error: WEEKLY recurrence duration cannot exceed 1 week"

Table 7-5: Reservation Errors

Description of Error	Server Log Error Code	Error Message
Reservation duration exceeds 1 hour and the frequency `FREQ` is set to `HOURLY` or the BY-rule is set to `BYHOUR` and occurs every hour, such as `BYHOUR=9,10`	15130	"pbs_rsub error: HOURLY recurrence duration cannot exceed 1 hour"
The PBS_TZID environment variable is not set correctly at the submission host; rejection at submission host	None	"pbs_rsub error: a valid `PBS_TZID` timezone environment variable is required"
The PBS_TZID environment variable is not set correctly at the submission host; rejection at Server	15135	"Unrecognized `PBS_TZID` environment variable"

7.8.9.3 Time Required Between Reservations

Leave enough time between reservations for the reservations and jobs in them to clean up. A job consumes resources even while it is in the `E` or exiting state. This can take longer when large files are being staged. If the job is still running when the reservation ends, it may take up to two minutes to be cleaned up. The reservation itself cannot finish cleaning up until its jobs are cleaned up. This will delay the start time of jobs in the next reservation unless there is enough time between the reservations for cleanup.

7.8.10 Reservation Information in the Accounting Log

The PBS Server writes an accounting record for each reservation in the job accounting file. The accounting record for a reservation is similar to that for a job. The accounting record for any job belonging to a reservation will include the reservation ID. See "Accounting Log" on page 475 of the PBS Professional Reference Guide.

7.8.11 Cannot Mix Reservations and mpp*

Do not request any mpp* resources in a reservation. PBS mpp* resources are loosely coupled to Cray resources, and those Cray resources are not completely controlled by PBS. A reservation requesting mppnodes, for example, does not prevent ALPS from running another job on those nodes. If this happens, the PBS job in the reservation is prevented from running, even though those resources are reserved. Mixing reservations and mpp* resources would lead to disappointment.

7.9 Dedicated Time

Dedicated time is one or more specific time periods defined by the administrator. These are not repeating time periods. Each one is individually defined.

During dedicated time, the only jobs PBS starts are those in special dedicated time queues. PBS schedules non-dedicated jobs so that they will not run over into dedicated time. Jobs in dedicated time queues are also scheduled so that they will not run over into non-dedicated time. PBS will attempt to backfill around the dedicated-non-dedicated time borders.

PBS uses walltime to schedule within and around dedicated time. If a job is submitted without a walltime to a non-dedicated-time queue, it will not be started until all dedicated time periods are over. If a job is submitted to a dedicated-time queue without a walltime, it will never run.

To submit a job to be run during dedicated time, use the -q <queue name> option to qsub and give the name of the dedicated-time queue you wish to use as the queue name. Queues are created by the administrator; see your administrator for queue name(s).

7.10 Using Comprehensive System Accounting

PBS supports Comprehensive System Accounting (CSA) on SGI Altix machines that are running SGI's ProPack 6 or 7 and have the Linux job container facility available. CSA provides accounting information about user jobs, called user job accounting.

CSA works the same with and without PBS. To run user job accounting, either the user must specify the file to which raw accounting information will be written, or an environment variable must be set. The environment variable is "ACCT_TMPDIR". This is the directory where a temporary file of raw accounting data is written.

To run user job accounting, the user issues the CSA command "`ja <filename>`" or, if the environment variable "ACCT_TMPDIR" is set, "`ja`". In order to have an accounting report produced, the user issues the command "`ja -<options>`" where the options specify that a report will be written and what kind. To end user job accounting, the user issues the command "`ja -t`"; the -t option can be included in the previous set of options. See the manpage on `ja` for details.

The starting and ending ja commands must be used before and after any other commands the user wishes to monitor. Here are examples of command line and a script:

On the command line:

```
qsub -N myjobname -l ncpus=1
    ja myrawfile
    sleep 50
    ja -c > myreport
    ja -t myrawfile
ctrl-D
```

Accounting data for the user's job (sleep 50) is written to myreport.

If the user creates a file foo with these commands:

```
#PBS -N myjobname
#PBS -l ncpus=1
ja myrawfile
sleep 50
ja -c > myreport
ja -t myrawfile
```

The user could run this script via qsub:

```
qsub foo
```

This does the same thing, via the script “foo”.

7.11 Running PBS in a UNIX DCE Environment

PBS Professional includes optional support for UNIX-based DCE. (By optional, we mean that the customer may acquire a copy of PBS Professional with the standard security and authentication module replaced with the DCE module.)

There are two -W options available with `qsub` which will enable a `dcelogin` context to be set up for the job when it eventually executes. The user may specify either an encrypted password or a forwardable/renewable Kerberos V5 TGT.

Specify the “`-W cred=dce`” option to `qsub` if a forwardable, renewable, Kerberos V5, TGT (ticket granting ticket) with the user as the listed principal is what is to be sent with the job. If the user has an established credentials cache and a non-expired, forwardable, renewable, TGT is in the cache, that information is used.

The other choice, “`-W cred=dce:pass`”, causes the `qsub` command to interact with the user to generate a DES encryption of the user's password. This encrypted password is sent to the PBS Server and MOM processes,

where it is placed in a job-specific file for later use by `pbs_mom` in acquiring a DCE login context for the job. The information is destroyed if the job terminates, is deleted, or aborts.

IMPORTANT:

The "`-W pwd=''`" option to `qsub` has been superseded by the above two options, and therefore should no longer be used.

Any acquired login contexts and accompanying DCE credential caches established for the job get removed on job termination or deletion.

```
qsub -Wcred=dce <other qsub options> job-script
```

IMPORTANT:

The "`-W cred`" option to `qsub` is not available under Windows.

7.12 Running PBS in a UNIX Kerberos Environment

PBS Professional includes optional support for Kerberos-only (i.e. no DCE) environment. (By optional, we mean that the customer may acquire a copy of PBS Professional with the standard security and authentication module replaced with the KRB5 module.) This is not supported under Windows.

To use a forwardable/renewable Kerberos V5 TGT specify the "`-W cred=krb5`" option to `qsub`. This will cause `qsub` to check the user's credential cache for a valid forwardable/renewable TGT which it will send to the Server and then eventually to the execution MOM. While it's at the Server and the MOM, this TGT will be periodically refreshed until either the job finishes or the maximum refresh time on the TGT is exceeded, whichever comes first. If the maximum refresh time on the TGT is exceeded, no KRB5 services will be available to the job, even though it will continue to run.

7.13 Support for Large Page Mode on AIX

A process running as part of a job can use large pages. The memory reported in `resources_used.mem` may be larger with large page sizes.

You can set an environment variable to request large memory pages:

```
LDR_CNTRL="LARGE_PAGE_DATA=M"
LDR_CNTRL="LARGE_PAGE_DATA=Y"
```

For more information see the man page for `setpcred`. This can be viewed with the command "man setpcred" on an AIX machine.

You can run a job that requests large page memory in "mandatory mode":

```
% qsub
export LDR_CNTRL="LARGE_PAGE_DATA=M"
/path/to/exe/bigprog
^D
```

You can run a job that requests large page memory in "advisory mode":

```
% qsub
export LDR_CNTRL="LARGE_PAGE_DATA=Y"
/path/to/exe/bigprog
^D
```

7.14 Checking License Availability

You can check to see where licenses are available. You can do either of the following:

- Display license information for the current host:

  ```
  qstat -Bf
  ```

- Display resources available (including licenses) on all hosts:

  ```
  qmgr
  Qmgr: print node @default
  ```

When looking at the server's license_count attribute, use the sum of the *Avail_Global* and *Avail_Local* values.

Chapter 8

Job Arrays

This chapter describes job arrays and their use. A job array represents a collection of jobs which only differ by a single index parameter. The purpose of a job array is twofold. It offers the user a mechanism for grouping related work, making it possible to submit, query, modify and display the set as a single unit. Second, it offers a way to possibly improve performance, because the batch system can use certain known aspects of the collection for speedup.

8.1 Definitions

Subjob

Individual entity within a job array (e.g. **1234[7]**, where **1234[]** is the job array itself, and **7** is the index) which has many properties of a job as well as additional semantics (defined below.)

Sequence_number

The numeric part of a job or job array identifier, e.g. **1234.**

Subjob index

The unique index which differentiates one subjob from another. This must be a non-negative integer.

Job array identifier

The identifier returned upon success when submitting a job array. The format is **sequence_number[]** or

sequence_number[].server.domain.com.

Job array range

A set of subjobs within a job array. When specifying a range, indices used must be valid members of the job array's indices.

8.1.1 Description

A job array is a compact representation of one or more jobs, called subjobs when part of a Job array, which have the same job script, and have the same values for all attributes and resources, with the following exceptions:

- each subjob has a unique index
- Job Identifiers of subjobs only differ by their indices
- the state of subjobs can differ

All subjobs within a job array have the same scheduling priority.

A job array is submitted through a single command which returns, on success, a "job array identifier" with a server-unique sequence number. Subjob indices are specified at submission time. These can be:

- a contiguous range, e.g. 1 through 100
- a range with a stepping factor, e.g. every second entry in 1 through 100 (1, 3, 5, ... 99)

A job array identifier can be used

- by itself to represent the set of all subjobs of the job array
- with a single index (a “job array identifier”) to represent a single subjob
- with a range (a “job array range”) to represent the subjobs designated by the range

8.1.2 Identifier Syntax

Job arrays have three identifier syntaxes:

- The job array object itself : 1234[].server or 1234[]
- A single subjob of a job array with index M: 1234[M].server or 1234[M]
- A range of subjobs of a job array: 1234[X-Y:Z].server or 1234[X-Y:Z]

Examples:

1234[].server.domain.com	Full job array identifier
1234[]	Short job array identifier
1234[73]	Subjob identifier of the 73rd index of job array 1234[]
1234	Error, if 1234[] is a job array
1234.server.domain.com	Error, if 1234[].server.domain.com is a job array

The sequence number (1234 in 1234[].server) is unique, so that jobs and job arrays cannot share a sequence number.

Note: Since some shells, for example csh and tcsh, read “[“ and “]” as shell metacharacters, job array names and subjob names will need to be enclosed in double quotes for all PBS commands.

Example:

```
qdel “1234.myhost[5]”
qdel “1234.myhost[]”
```

Single quotes will work, except where you are using shell variable substitution.

8.2 qsub: Submitting a Job Array

To submit a job array, qsub is used with the option **-J range**, where **range** is of the form **X-Y[:Z]**. **X** is the starting index, **Y** is the ending index, and **Z** is the optional **stepping factor**. X and Y must be whole numbers, and Z must be a positive integer. Y must be greater than X. If Y is not a multiple of the stepping factor above X, (i.e. it won't be used as an index value) the highest index used will be the next below Y. For example, 1-100:2 gives 1, 3, 5, ... 99.

Blocking qsub waits until the entire job array is complete, then returns the exit status of the job array.

Interactive submission of job arrays is not allowed.

Examples:

Example 1: To submit a job array of 10,000 subjobs, with indices 1, 2, 3, ... 10000:

```
$ qsub -J 1-10000 job.scr
1234[].server.domain.com
```

Example 2: To submit a job array of 500 subjobs, with indices 500, 501, 502, ... 1000:

```
$ qsub -J 500-1000 job.scr
1235[].server.domain.com
```

Example 3: To submit a job array with indices 1, 3, 5 ... 999:

```
$ qsub -J 1-1000:2 job.scr
1236[].server.domain.com
```

8.2.1 Interactive Job Submission

Job arrays do not support interactive submission.

8.3 Job Array Attributes

Job arrays and subjobs have all of the attributes of a job. In addition, they have the following when appropriate. These attributes are read-only.

Table 8-1: Job Array Attributes

Name	Type	Applies To	Value
array	boolean	job array	True if item is job array
array_id	string	subjob	Subjob's job array identifier
array_index	string	subjob	Subjob's index number
array_state_count	string	job array	Similar to state_count attribute for server and queue objects. Lists number of subjobs in each state.
array_indices_remaining	string	job array	List of indices of subjobs still queued. Range or list of ranges, e.g. 500, 552, 596-1000
array_indices_submitted	string	job array	Complete list of indices of subjobs given at submission time. Given as range, e.g. 1-100

8.4 Job Array States

See “Job Array States” on page 466 of the PBS Professional Reference Guide and “Subjob States” on page 467 of the PBS Professional Reference Guide.

8.5 PBS Environmental Variables

Table 8-2: PBS Environmental Variables

Environment Variable Name	Used For	Description
$PBS_ARRAY_INDEX	subjobs	Subjob index in job array, e.g. 7
$PBS_ARRAY_ID	subjobs	Identifier for a job array. Sequence number of job array, e.g. 1234[].server
$PBS_JOBID	Jobs, subjobs	Identifier for a job or a subjob. For subjob, sequence number and subjob index in brackets, e.g. 1234[7].server

8.6 File Staging

File staging for job arrays is like that for jobs, with an added variable to specify the subjob index. This variable is **^array_index^**. This is the name of the variable that will be used for the actual array index. The stdout and stderr files follow the naming convention for jobs, but include the identifier of the job array, which includes the subscripted index. As with jobs, the stagein and stageout keywords require the -W option to qsub.

8.6.1 Specifying Files To Be Staged In or Staged Out

You can specify files to be staged in before the job runs and staged out after the job runs by using `-W stagein=file_list` and `-W stageout=file_list`. You can use these as options to qsub, or as directives in the job script.

The *file_list* takes the form:

execution_path@hostname:storage_path[,...]

for both stagein and stageout.

The name *execution_path* is the name of the file in the job's staging and execution directory (on the execution host). The `execution_path` can be relative to the job's staging and execution directory, or it can be an absolute path.

The '@' character separates the local specification from the remote specification.

The name *storage_path* is the file name on the host specified by hostname. For stagein, this is the location where the input files come from. For stageout, this is where the output files end up when the job is done. You must specify a hostname. The name can be absolute, or it can be relative to the user's home directory on the remote machine.

IMPORTANT:

It is advisable to use an absolute pathname for the storage_path. Remember that the path to your home directory may be different on each machine, and that when using `sandbox = PRIVATE,` you may or may not have a home directory on all execution machines.

For stagein, the direction of travel is **from** storage_path **to** execution_path.

For stageout, the direction of travel is **from** execution_path **to** storage_path.

When staging more than one filename, separate the filenames with a comma and enclose the entire list in double quotes.

Examples:

storage_path: store:/film

Data files used as input: frame1, frame2, frame3

execution_path: pix

Executable: a.out

For this example, `a.out` produces `frame2.out` from `frame2.`

```
#PBS -W stagein=pix/in/frame^array_index^@store:/film/
   frame^array_index^
#PBS- W stageout=pix/out/frame^array_index^.out
   @store:/film/frame^array_index^.out
#PBS -J 1-3 a.out frame$PBS_ARRAY_INDEX ./in ./out
```

Note that the stageout statement is all one line, broken here for readability.

The result will be that the user's directory named "film" contains the original files `frame1, frame2, frame3`, plus the new files `frame1.out, frame2.out` and `frame3.out`.

8.6.1.1 Scripts

Example 1: In this example, we have a script named ArrayScript which calls scriptlet1 and scriptlet2.

All three scripts are located in `/homedir/testdir`.

```
#!/bin/sh
#PBS -N ArrayExample
#PBS -J 1-2
echo "Main script: index " $PBS_ARRAY_INDEX
/homedir/testdir/scriptlet$PBS_ARRAY_INDEX
```

In our example, scriptlet1 and scriptlet2 simply echo their names. We run ArrayScript using the qsub command:

```
qsub ArrayScript
```

Example 2: In this example, we have a script called `StageScript`. It takes two input files, `dataX` and `extraX`, and makes an output file,

newdataX, as well as echoing which iteration it is on. The dataX and extraX files will be staged from inputs to work, then newdataX will be staged from work to outputs.

```
#!/bin/sh
#PBS -N StagingExample
#PBS -J 1-2
#PBS -W stagein="/homedir/work/data^array_index^
   @host1:/homedir/inputs/data^array_index^, \
   /homedir/work/extra^array_index^ \
   @host1:/homedir/inputs/extra^array_index^"
#PBS -W stageout=/homedir/work/newdata^array_index^
   @host1:/homedir/outputs/newdata^array_index^
echo "Main script: index " $PBS_ARRAY_INDEX
cd /homedir/work
cat data$PBS_ARRAY_INDEX extra$PBS_ARRAY_INDEX \
   >> newdata$PBS_ARRAY_INDEX
```

Local path (execution directory):

```
/homedir/work
```

Remote host (data storage host):

```
host1
```

Remote path for inputs (original data files dataX and extraX):

```
/homedir/inputs
```

Remote path for results (output of computation newdataX):

```
/homedir/outputs
```

StageScript resides in /homedir/testdir. In that directory, we can run it by typing:

```
qsub StageScript
```

It will run in /homedir, our home directory, which is why the line

```
"cd /homedir/work"
```

is in the script.

Example 3: In this example, we have the same script as before, but we will run it in a staging and execution directory created by PBS. StageScript takes two input files, `dataX` and `extraX`, and makes an output file, `newdataX`, as well as echoing which iteration it is on. The `dataX` and `extraX` files will be staged from `inputs` to the staging and execution directory, then `newdataX` will be staged from the staging and execution directory to `outputs`.

```
#!/bin/sh
#PBS -N StagingExample
#PBS -J 1-2
#PBS -W stagein="data^array_index^\
   @host1:/homedir/inputs/data^array_index^, \
   extra^array_index^ \
   @host1:/homedir/inputs/extra^array_index^"
#PBS -W stageout=newdata^array_index^\
   @host1:/homedir/outputs/newdata^array_index^
echo "Main script: index " $PBS_ARRAY_INDEX
cat data$PBS_ARRAY_INDEX extra$PBS_ARRAY_INDEX \
   >> newdata$PBS_ARRAY_INDEX
```

Local path (execution directory):

created by PBS; we don't know the name

Remote host (data storage host):

`host1`

Remote path for inputs (original data files `dataX` and `extraX`):

`/homedir/inputs`

Remote path for results (output of computation `newdataX`):

`/homedir/outputs`

`StageScript` resides in `/homedir/testdir`. In that directory, we can run it by typing:

```
qsub StageScript
```

It will run in the staging and execution directory created by PBS. See section 7.6 “Input/Output File Staging” on page 166.

8.6.1.2 Output Filenames

The name of the job array will default to the script name if no name is given via qsub -N.

For example, if the sequence number were 1234,

```
#PBS -N fixgamma
```

would give stdout for index number 7 the name fixgamma.o1234.7 and stderr the name fixgamma.e1234.7. The name of the job array can also be given through stdin.

8.6.2 Job Array Staging Syntax on Windows

In Windows the stagein and stageout string must be contained in double quotes when using ^array_index^.

Example of a stagein:

```
qsub -W stagein="foo.^array_index^@host-
   1:C:\WINNT\Temp\foo.^array_index^" -J 1-5
   stage_script
```

Example of a stageout:

```
qsub -W stageut="C:\WINNT\Temp\foo.^array_index^@host-
   1:Q:\my_username\foo. ^array_index^_out" -J 1-5
   stage_script
```

8.7 PBS Commands

8.7.1 PBS Commands Taking Job Arrays as Arguments

Note: Some shells such as csh and tcsh use the square bracket ("[", "]") as a metacharacter. When using one of these shells, and a PBS command taking subjobs, job arrays or job array ranges as arguments, the subjob, job array or job array range must be enclosed in double quotes.

The following table shows PBS commands that take job arrays, subjobs or ranges as arguments. The cells in the table indicate which objects are acted upon. In the table,

Array[] =	the job array object
Array[Range] =	the set of subjobs of the job array with indices in range given
Array[Index] =	the individual subjob of the job array with the index given
Array[RUNNING] =	the set of subjobs of the job array which are currently running
Array[QUEUED] =	the set of subjobs of the job array which are currently queued
Array[REMAINING] =	the set of subjobs of the job array which are queued or running
Array[DONE]=	the set of subjobs of the job array which have finished running

Table 8-3: PBS Commands Taking Job Arrays as Arguments

	Argument to Command		
Command	Array[]	Array[Range]	Array[Index]
`qstat`	Array[]	Array[Range]	Array[Index]
`qdel`	Array[] & Array[REMAINING]	Array[Range] where Array[REMAINING]	Array[Index]
`qalter`	Array[]	erroneous	erroneous
`qorder`	Array[]	erroneous	erroneous
`qmove`	Array[] & Array[QUEUED]	erroneous	erroneous
`qhold`	Array[] & Array[QUEUED]	erroneous	erroneous
`qrls`	Array[] & Array[QUEUED]	erroneous	erroneous
`qrerun`	Array[RUNNING] & Array[DONE]	Array[Range] where Array[RUNNING]	Array[Index]
`qrun`	erroneous	Array[Range] where Array[QUEUED]	Array[Index]
`tracejob`	erroneous	erroneous	Array[Index]
`qsig`	Array[RUNNING]	Array[Range] where Array[RUNNING]	Array[Index]
`qmsg`	erroneous	erroneous	erroneous

8.7.2 qstat: Status of a Job Array

The qstat command is used to query the status of a Job Array. The default output is to list the Job Array in a single line, showing the Job Array Identifier. Options can be combined. To show the state of all running subjobs, use -t -r. To show the state only of subjobs, not job arrays, use -t -J.

Table 8-4: Job Array and Subjob Options to qstat

Option	Result
-t	Shows state of job array object and subjobs. Will also show state of jobs.
-J	Shows state only of job arrays.
-p	Prints the default display, with column for Percentage Completed. For a job array, this is the number of subjobs completed or deleted divided by the total number of subjobs. For a job, it is time used divided by time requested.

Examples:

We run an example job and an example job array, on a machine with 2 processors:

demoscript:

```
#!/bin/sh
#PBS -N JobExample
sleep 60
```

arrayscript:

```
#!/bin/sh
#PBS -N ArrayExample
#PBS -J 1-5
sleep 60
```

We run these scripts using qsub.

```
qsub arrayscript
1235[].host
qsub demoscript
1236.host
```

Then:

```
qstat
Job id      Name         User        Time Use S Queue
----------- ------------ ----------  -------- - -----
1235[].host ArrayExample user1              0 B workq
1236.host   JobExample   user1              0 Q workq

qstat -J

Job id      Name         User        Time Use S Queue
----------- ------------ ----------  -------- - -----
1235[].host ArrayExample user1              0 B workq

qstat -p

Job id      Name         User        % done  S Queue
----------- ------------ ----------  ------- - -----
1235[].host ArrayExample user1           0  B workq
1236.host   JobExample   user1          --  Q workq

qstat -t

Job id       Name         User        Time Use S Queue
-----------  ------------ ----------  -------- - -----
1235[].host  ArrayExample user1              0 B workq
1235[1].host ArrayExample user1       00:00:00 R workq
```

```
1235[2].host ArrayExample user1       00:00:00 R workq
1235[3].host ArrayExample user1              0 Q workq
1235[4].host ArrayExample user1              0 Q workq
1235[5].host ArrayExample user1              0 Q workq
1236.host    JobExample   user1              0 Q workq
```

qstat -Jt

```
Job id       Name         User   Time Use S Queue
------------ ------------ -----  -------- - -----
1235[1].host ArrayExample user1  00:00:00 R workq
1235[2].host ArrayExample user1  00:00:00 R workq
1235[3].host ArrayExample user1         0 Q workq
1235[4].host ArrayExample user1         0 Q workq
1235[5].host ArrayExample user1         0 Q workq
```

After the first two subjobs finish:

qstat -Jtp

```
Job id       Name         User   % done S Queue
------------ ------------ -----  ------ - -----
1235[1].host ArrayExample user1     100 X workq
1235[2].host ArrayExample user1     100 X workq
1235[3].host ArrayExample user1      -- R workq
1235[4].host ArrayExample user1      -- R workq
1235[5].host ArrayExample user1      -- Q workq
```

qstat -pt

```
Job id       Name         User   % done S Queue
------------ ------------ -----  ------ - -----
1235[].host  ArrayExample user1      40 B workq
```

```
1235[1].host ArrayExample user1     100 X workq
1235[2].host ArrayExample user1     100 X workq
1235[3].host ArrayExample user1      -- R workq
1235[4].host ArrayExample user1      -- R workq
1235[5].host ArrayExample user1      -- Q workq
1236.host    JobExample   user1      -- Q workq
```

Now if we wait until only the last subjob is still running:

qstat -rt

```
                                                    Req'd  Req'd   Elap
Job ID     Username Queue Jobname   SessID NDS TSK Memory Time S Time
---------- ------  -----  --------  ---- --- --- ------ ---- - -----
1235[5].host user1  workq ArrayExamp 3048  --   1    --    --  R 00:00
1236.host    user1  workq JobExample 3042  --   1    --    --  R 00:00
```

qstat -Jrt

```
                                                  Req'd  Req'd   Elap
Job ID     Username Queue Jobname  SessID NDS TSK Memory Time S Time
---------- -------- ----- --------- ---- --- --- ------ ---- - -----
1235[5].host   user1 workq ArrayExamp 048 --   1    --    --  R 00:01
```

8.7.3 qdel: Deleting a Job Array

The `qdel` command will take a job array identifier, subjob identifier or job array range. The indicated object(s) are deleted, including any currently running subjobs. Running subjobs are treated like running jobs. Subjobs not running will be deleted and never run. Only one email is sent per deleted job array, so deleting a job array of 5000 subjobs results in one email being sent.

8.7.4 qalter: Altering a Job Array

The qalter command can only be used on a job array object, not on subjobs or ranges. Job array attributes are the same as for jobs.

8.7.5 qorder: Ordering Job Arrays in the Queue

The qorder command can only be used with job array objects, not on subjobs or ranges. This will change the queue order of the job array in association with other jobs or job arrays in the queue.

8.7.6 qmove: Moving a Job Array

The qmove command can only be used with job array objects, not with subjobs or ranges. Job arrays can only be moved from one server to another if they are in the 'Q', 'H', or 'W' states, and only if there are no running subjobs. The state of the job array object is preserved in the move. The job array will run to completion on the new server.

As with jobs, a qstat on the server from which the job array was moved will not show the job array. A qstat on the job array object will be redirected to the new server.

Note: The subjob accounting records will be split between the two servers.

8.7.7 qhold: Holding a Job Array

The qhold command can only be used with job array objects, not with subjobs or ranges. A hold can be applied to a job array only from the 'Q', 'B' or 'W' states. This will put the job array in the 'H', held, state. If any subjobs are running, they will run to completion. No queued subjobs will be started while in the 'H' state.

8.7.8 qrls: Releasing a Job Array

The qrls command can only be used with job array objects, not with subjobs or ranges. If the job array was in the 'Q' or 'B' state, it will be returned to that state. If it was in the 'W' state, it will be returned to that state unless its waiting time was reached, it will go to the 'Q' state.

8.7.9 qrerun: Requeueing a Job Array

The qrerun command will take a job array identifier, subjob identifier or job array range. If a job array identifier is given as an argument, it is returned to its initial state at submission time, or to its altered state if it has been qaltered. All of that job array's subjobs are requeued, which includes those that are currently running, and completed and deleted. If a subjob or range is given, those subjobs are requeued as jobs would be.

8.7.10 qrun: Running a Job Array

The qrun command takes a subjob or a range of subjobs, not a job array object. If a single subjob is given as the argument, it is run as a job would be. If a range of subjobs is given as the argument, the non-running subjobs within that range will be run.

8.7.11 tracejob on Job Arrays

The tracejob command can be run on job arrays and individual subjobs. When tracejob is run on a job array or a subjob, the same information is displayed as for a job, with additional information for a job array. Note that subjobs do not exist until they are running, so tracejob will not show any information until they are. When tracejob is run on a job array, the information displayed is only that for the job array object, not the subjobs. Job arrays themselves do not produce any MOM log information. Running tracejob on a job array will give information about why a subjob did not start.

8.7.12 qsig: Signaling a Job Array

If a job array object, subjob or job array range is given to qsig, all currently running subjobs within the specified set will be sent the signal.

8.7.13 qmsg: Sending Messages

The qmsg command is not supported for job arrays.

8.8 Other PBS Commands Supported for Job Arrays

8.8.1 qselect: Selection of Job Arrays

The default behavior of qselect is to return the job array identifier, without returning subjob identifiers.

Note: qselect will not return any job arrays when the state selection (-s) option restricts the set to ‘R’, ‘S’, ‘T’ or ‘U’, because a job array will never be in any of these states. However, qselect can be used to return a list of subjobs by using the -t option.

Options to qselect can be combined. For example, to restrict the selection to subjobs, use both the -J and the -T options. To select only running subjobs, use -J -T -sR.

Table 8-5: Options to qselect for Job Arrays

Option	Selects	Result
(none)	jobs, job arrays	Shows job and job array identifiers
-J	job arrays	Shows only job array identifiers
-T	jobs, subjobs	Shows job and subjob identifiers

8.9 Job Arrays and xpbs

xpbs does not support job arrays.

8.10 More on Job Arrays

8.10.1 Job Array Run Limits

Jobs and subjobs are treated the same way by job run limits. For example, if *max_user_run* is set to 5, a user can have a maximum of 5 subjobs and/or jobs running.

8.10.2 Starving

A job array's starving status is based on the queued portion of the array. This means that if there is a queued subjob which is starving, the job array is starving. A running subjob retains its starving status when it was started.

8.10.3 Job Array Dependencies

Job dependencies are supported:

between job arrays and job arrays

between job arrays and jobs

between jobs and job arrays

Note: Job dependencies are not supported for subjobs or ranges of subjobs.

8.10.4 The "Rerunnable" Flag and Job Arrays

Job arrays are required to be rerunnable. PBS will not accept a job array that is not marked as rerunnable. You can submit a job array without specifying whether it is rerunnable, and PBS will automatically mark it as rerunnable.

8.10.5 Accounting

Job accounting records for job arrays and subjobs are the same as for jobs. When a job array has been moved from one server to another, the subjob accounting records are split between the two servers, except that there will be no 'Q' records for subjobs.

8.10.6 Checkpointing

Checkpointing is not supported for job arrays. On systems that support checkpointing, subjobs are not checkpointed, instead they run to completion.

8.10.7 Prologues and Epilogues

If defined, prologues and epilogues will run at the beginning and end of each subjob, but not for job arrays.

8.10.8 Job Array Exit Status

The exit status of a job array is determined by the status of each of the completed subjobs. It is only available when all valid subjobs have completed. The individual exit status of a completed subjob is passed to the epilogue, and is available in the 'E' accounting log record of that subjob.

Table 8-6: Job Array Exit Status

Exit Status	Meaning
0	All subjobs of the job array returned an exit status of 0. No PBS error occurred. Deleted subjobs are not considered
1	At least 1 subjob returned a non-zero exit status. No PBS error occurred.
2	A PBS error occurred.

8.10.9 Scheduling Job Arrays

All subjobs within a job array have the same scheduling priority.

8.10.9.1 Preemption

Individual subjobs may be preempted by higher priority work.

8.10.9.2 Peer Scheduling

Peer scheduling does not support job arrays.

8.10.9.3 Fairshare

Subjobs are treated like jobs with respect to fairshare ordering, fairshare accounting and fairshare limits. If running enough subjobs of a job array causes the priority of the owning entity to change, additional subjobs from that job array may not be the next to start.

8.10.9.4 Placement Sets and Node Grouping

All nodes associated with a single subjob should belong to the same placement set or node group. Different subjobs can be put on different placement sets or node groups.

8.11 Job Array Caveats

8.11.1 Job Arrays Are Rerunnable

Job arrays are required to be rerunnable, and are rerunnable by default.

8.11.2 Mail for Job Arrays

The `-m` and `-M qsub` options are ignored for job arrays. No status notifications are mailed for job arrays.

When stagein or stageout fails for a job array, PBS sends mail to the job owner.

Chapter 9

Multiprocessor Jobs

9.1 Job Placement

Placement sets allow partitioning by multiple resources, so that a vnode may be in one set that share a value for one resource, and another set that share a different value for a different resource. See the **PBS Professional Administrator's Guide**.

If a job requests grouping by a resource, i.e. *place=group=resource*, then the chunks are placed as requested and complex-wide node grouping is ignored.

If a job is to use node grouping but the required number of vnodes is not defined in any one group, grouping is ignored. This behavior is unchanged.

9.2 Submitting SMP Jobs

To submit a job which should run on one host and which requires a certain number of cpus and amount of memory, submit the job with:

```
qsub -l select=ncpus=N:mem=M -l place=group=host
```

When the job is run, the PBS_NODEFILE will contain one entry, the name of the selected execution host. Generally this is ignored for SMP jobs as all processes in the job are run on the host where the job script is run. The job will have two environment variables, NCPUS and OMP_NUM_THREADS, set to N, the number of CPUs allocated.

9.3 Submitting MPI Jobs

The preferred method for submitting an MPI job is by specifying one chunk per MPI task. For example, for a 10-way MPI job with 2gb of memory per MPI task, you would use:

```
qsub -l select=10:ncpus=1:mem=2gb
```

If you have a cluster of small systems with for example 2 CPUs each, and you wish to submit an MPI job that will run on four separate hosts, then submit:

```
qsub -l select=4:ncpus=1 -l place=scatter
```

The PBS_NODEFILE file will contain one entry for each of the hosts allocated to the job. In the example above, it would contain 4 lines. The variables NCPUS and OMP_NUM_THREADS will be set to one.

If you do not care where the four MPI processes are run, you may submit:

```
qsub -l select=4:ncpus=1 -l place=free
```

and the job will run on 2, 3, or 4 hosts depending on what is available.

For this example, PBS_NODEFILE will contain 4 entries, either four separate hosts, or 3 hosts one of which is repeated once, or 2 hosts, etc. NCPUS and OMP_NUM_THREADS will be set 1 or 2 depending on the number of cpus allocated from the first listed host.

9.3.1 The mpiprocs Resource

The number of MPI processes for a job is controlled by the value of the resource mpiprocs. The mpiprocs resource controls the contents of the PBS_NODEFILE on the host which executes the top PBS task for the PBS job (the one executing the PBS job script.) See “Built-in Resources” on page 350 of the PBS Professional Reference Guide. The PBS_NODEFILE contains one line per MPI process with the name of the host on which that process should execute. The number of lines in PBS_NODEFILE is equal to the sum of the values of mpiprocs over all chunks requested by the job. For each chunk with mpiprocs=P, (where P > 0), the host name (the value of the allocated vnode's resources_available.host) is written to the PBS_NODEFILE exactly P times.

If a user wishes to run two MPI processes on each of 3 hosts and have them "share" a single processor on each host, the user would request

```
-lselect=3:ncpus=1:mpiprocs=2
```

The PBS_NODEFILE would contain the following list:

```
VnodeA
VnodeA
VnodeB
VnodeB
VnodeC
VnodeC
```

If you want 3 chunks, each with 2 CPUs and running 2 MPI process, use:

```
-l select=3:ncpus=2:mpiprocs=2...
```

The PBS_NODEFILE would contain the following list:

```
VnodeA
VnodeA
VnodeB
VnodeB
VnodeC
VnodeC
```

9.4 OpenMP Jobs with PBS

PBS Professional supports OpenMP applications by setting the OMP_NUM_THREADS variable automatically based on the resource request of a job in the environment of the job. The OpenMP run-time will pick up the value of OMP_NUM_THREADS and create threads appropriately.

The OMP_NUM_THREADS value can be set explicitly by using the ompthreads pseudo-resource for any chunk within the select statement. If ompthreads is not used, then OMP_NUM_THREADS is set to the value of the ncpus resource of that chunk. If neither ncpus nor ompthreads is used within the select statement, then OMP_NUM_THREADS is set to 1.

To submit an OpenMP job is as a single chunk, for a 2-CPU job requiring 10gb of memory, you would use:

```
qsub -l select=1:ncpus=2:mem=10gb
```

You might be running an OpenMP application on a host and wish to run fewer threads than the number of CPUs requested. This might be because the threads need exclusive access to shared resources in a multi-core processor system, such as to a cache shared between cores, or to the memory shared between cores. If you want one chunk, with 16 CPUs and 8 threads:

```
qsub -l select=1:ncpus=16:ompthreads=8
```

You might be running an OpenMP application on a host and wish to run more threads than the number of CPUs requested (because each thread is I/O bound perhaps). If you want one chunk, with eight CPUs and 16 threads:

```
qsub -l select=1:ncpus=8:ompthreads=16
```

9.5 Hybrid MPI-OpenMP Jobs

For jobs that are both MPI and multi-threaded, the number of threads per chunk, for all chunks, is set to the number of threads requested (explicitly or implicitly) in the first chunk, except for MPIs that have been integrated with the PBS TM API. For these MPIs (LAM MPI), you can specify the

number of threads separately for each chunk. This means that for most MPIs, OMP_NUM_THREADS and NCPUS will default to the number of ncpus requested on the first chunk, and for integrated MPIs, you can set the ompthreads resource separately for each chunk.

Should you have a job that is both MPI and multi-threaded, you can request one chunk for each MPI process, or set mpiprocs to the number of MPI processes you want on each chunk.

For example, to request 4 chunks, each with 1 MPI process, 2 CPUs and 2 threads:

```
qsub -l select=4:ncpus=2
```

or

```
qsub -l select=4:ncpus=2:ompthreads=2
```

To request 4 chunks, each with 2 CPUs and 4 threads:

```
qsub -l select=4:ncpus=2:ompthreads=4
```

To request 16 MPI processes each with 2 threads on machines with 2 processors:

```
qsub -l select=16:ncpus=2
```

To request two chunks, each with 8 CPUs and 8 MPI tasks and four threads:

```
qsub -l select=2:ncpus=8:mpiprocs=8:ompthreads=4
```

Example:

```
qsub -l select=4:ncpus=2
```

This request is satisfied by 4 CPUs from VnodeA, 2 from VnodeB and 2 from VnodeC, so the following is written to the PBS_NODEFILE:

```
VnodeA
VnodeA
VnodeB
VnodeC
```

The OpenMP environment variables are set (for the 4 PBS tasks corresponding to the 4 MPI processes) as follows:

- For PBS task #1 on VnodeA: OMP_NUM_THREADS=2 NCPUS=2
- For PBS task #2 on VnodeA: OMP_NUM_THREADS=2 NCPUS=2
- For PBS task #3 on VnodeB: OMP_NUM_THREADS=2 NCPUS=2
- For PBS task #4 on VnodeC: OMP_NUM_THREADS=2 NCPUS=2

Example:

```
qsub -l select=3:ncpus=2:mpiprocs=2:ompthreads=1
```

This is satisfied by 2 CPUs from each of three vnodes (VnodeA, VnodeB, and VnodeC), so the following is written to the PBS_VNODEFILE:

```
VnodeA
VnodeA
VnodeB
VnodeB
VnodeC
VnodeC
```

The OpenMP environment variables are set (for the 6 PBS tasks corresponding to the 6 MPI processes) as follows:

- For PBS task #1 on VnodeA: OMP_NUM_THREADS=1 NCPUS=1
- For PBS task #2 on VnodeA: OMP_NUM_THREADS=1 NCPUS=1
- For PBS task #3 on VnodeB: OMP_NUM_THREADS=1 NCPUS=1
- For PBS task #4 on VnodeB: OMP_NUM_THREADS=1 NCPUS=1
- For PBS task #5 on VnodeC: OMP_NUM_THREADS=1 NCPUS=1
- For PBS task #6 on VnodeC: OMP_NUM_THREADS=1 NCPUS=1

To run two threads on each of N chunks, each running a process, all on the same Altix:

```
qsub -l select=N:ncpus=2 -l place=pack
```

This starts N processes on a single host, with two OpenMP threads per process, because OMP_NUM_THREADS=2.

9.6 MPI Jobs with PBS

PBS creates one MPI process per chunk.

For most implementations of the Message Passing Interface (MPI), you would use the `mpirun` command to launch your application. For example, here is a sample PBS script for an MPI job:

```
#PBS -l select=arch=linux
#
mpirun -np 32 -machinefile $PBS_NODEFILE a.out
```

9.6.1 MPICH Jobs With PBS

For users of PBS with MPICH on Linux, the `mpirun` command has been changed slightly. The syntax and arguments are the same except for one option, which should not be set by the user:

-machinefile file
: PBS supplies the machinefile. If the user tries to specify it, PBS will print a warning that it is replacing the machinefile.

Example of using mpirun:

```
#PBS -l select=arch=linux
#
mpirun a.out
```

Under Windows the `-localroot` option to `MPICH`'s `mpirun` command may be needed in order to allow the job's processes to run more efficiently.

9.6.2 MPI Jobs Using LAM MPI

The `pbs_mpilam` command follows the convention of LAM's `mpirun`. The "nodes" here are LAM nodes. LAM's `mpirun` has two syntax forms:

pbs_mpilam/mpirun [global_options] [<where>] <program> [--args]

pbs_mpilam/mpirun [global_options] <schema file>

Where

<where> is a set of node and/or CPU identifiers indicating where to start <program>:

Nodes: n<list>, e.g., n0-3,5

CPUS: c<list>, e.g., c0-3,5

Extras: h (local node), o (origin node), N (all nodes), C (all CPUs)

<schema file> is an ASCII file containing a description of the programs which constitute an application.

The first form is fully supported by PBS: all user MPI processes are tracked. The second form is supported, but user MPI processes are not tracked.

CAUTION: Keep in mind that if the <where> argument and global option -np or -c are not specified in the command line, then pbs_mpilam will expect an ASCII schema file as argument.

9.6.3 MPI Jobs Using AIX, POE

PBS users of AIX machines running IBM's Parallel Operating Environment, or POE, can run jobs on the HPS using either IP or US mode. PBS will manage the HPS. PBS can track the resources for MPI, LAPI programs or a mix of MPI and LAPI programs. LoadLeveler is not required in order to use InfiniBand switches in User Space mode. PBS works with a standard InfiniBand configuration. Any job that can run under IBM poe can run under PBS, with the exceptions and differences noted here.

Under PBS, the poe command is slightly different. See "IBM's poe: pbsrun.poe" on page 142 of the PBS Professional Reference Guide.

9.6.3.1 The User's Environment

In order to ensure that the InfiniBand switch can be used for a job, the job must have PBS_GET_IBWINS = 1 in its environment. This can be handled either by the administrator or by the job submitter. Users submitting poe jobs may choose to set PBS_GET_IBWINS = 1 in their shell environment, and use the -V option to the qsub command:

- csh:

```
setenv PBS_GET_IBWINS 1
```

- bash:

```
PBS_GET_IBWINS = 1
export PBS_GET_IBWINS
```

PBS requires no other changes to the user's environment.

Do not set the PBS_O_HOST environment variable. See section 9.6.3.5.3 "Environment Variable" on page 240.

9.6.3.2 Using the InfiniBand Switch

To ensure that a job uses the InfiniBand switch, make sure that the job's environment has PBS_GET_IBWINS set to 1. This can be accomplished the following ways:

- The administrator sets this value for all jobs.
- To set the environment variable for each job, the job submitter sets PBS_GET_IBWINS = 1 in their shell environment, and uses the -V option to every qsub command. See the previous section.
- To set the environment variable for one job, the job submitter uses the "-v PBS_GET_IBWINS = 1" option to the qsub command.

9.6.3.3 Restrictions on poe Jobs

Users submitting poe jobs can run poe outside of PBS, but they will see this warning:

```
pbsrun.poe: Warning, not running under PBS
```

Users cannot run poe jobs without arguments inside PBS. Attempting to do this will give the following error:

```
pbsrun.poe: Error, interactive program name entry not
   supported under PBS
```

poe will exit with a value of 1.

Some environment variables and options to poe behave differently under PBS. These differences are described in the next section.

9.6.3.4 Options to poe and Environment Variables

The usage for poe is:

poe [program] [program_options] [poe options]

Users submitting jobs to poe can set environment variables instead of using options to poe. The equivalent environment variable is listed with its poe option. All options and environment variables except the following are passed to poe:

-devtype, MP_DEVTYPE
: If InfiniBand is not specified in either the option or the environment variable, US mode is not used for the job.

-euidevice, MP_EUIDEVICE
: Ignored by PBS.

-euilib {ip|us}, MP_EUILIB
: If set to *us*, the job runs in User Space mode.

: If set to any other value, that value is passed to IBM poe.

: If the command line option -euilib is set, it will take precedence over the MP_EUILIB environment variable.

-hostfile, -hfile, MP_HOSTFILE
: Ignored. If this is specified, PBS prints the following:

```
pbsrun.poe: Warning, -hostfile value replaced
   by PBS
```

: or

```
pbsrun.poe: Warning -hfile value replaced by
   PBS
```

If this environment variable is set when a poe job is submitted, PBS prints the following error message:

```
pbsrun.poe: Warning MP_HOSTFILE value replaced
  by PBS
```

-instances, MP_INSTANCES

The option and the environment variable are treated differently:

-instances

If the option is set, PBS prints a warning:

```
pbsrun.poe: Warning, -instances cmd line option
  removed by PBS
```

MP_INSTANCES

If the environment variable is set, PBS uses it to calculate the number of network windows for the job.

The maximum value allowed can be requested by using the string "max" for the environment variable.

If the environment variable is set to a value greater than the maximum allowed value, it is replaced with the maximum allowed value.

The default maximum value is *4*.

-procs, MP_PROCS

This option or environment variable should be set to the total number of mpiprocs requested by the job when using US mode.

If neither this option nor the MP_PROCS environment variable is set, PBS uses the number of entries in $PBS_NODEFILE.

If this option is set to *N*, and the job is submitted with a total of *M* mpiprocs:

If *N* >=*M*: The value *N* is passed to IBM poe.

If *N* < *M* and US mode is not being used: The value *N* is passed to poe.

If $N < M$ and US mode is being used: US mode is turned off and a warning is printed:

```
pbsrun.poe: Warning, user mode disabled due to
  MP_PROCS setting
```

9.6.3.5 Caveats

9.6.3.5.1 Multi-host Jobs

If you wish to run a multi-host job, it must not run on a mix of InfiniBand and non-InfiniBand hosts. It can run entirely on hosts that are non-InfiniBand., or on hosts that are all using InfiniBand, but not both.

9.6.3.5.2 Job Submission Format

Do not submit InfiniBand jobs in which the select statement specifies only a number, for example:

```
$ export PBS_GET_IBWINS=1
$ qsub -koe -mn -l select=1 -V jobname
```

Instead, use the equivalent request which specifies a resource:

```
$ export PBS_GET_IBWINS=1
$ qsub -koe -mn -l select=1:ncpus=1 -V jobname
```

9.6.3.5.3 Environment Variable

Do not set the PBS_O_HOST environment variable. Using the `qsub` command with the -V option will fail.

9.6.3.5.4 If Your Complex Contains Machines Not on the HPS

If your complex contains machines that are not on the HPS, and you wish to run on the HPS, you must specify machines on the HPS. Your administrator will define a resource on each host on the HPS. To specify machines on the HPS, you must request the "hps" resource in your select statement. For this example, the resource is “hps”.

Using place=scatter: When "scatter" is used, the 4 chunks are on different hosts so each host has 1 hps resource:

```
% qsub -l select=4:ncpus=2:hps=1
```

Using place=pack: When "pack" is used, all the chunks are put on one host so a chunk with no resources and one "hps" must be specified:

```
% qsub -l select=4:ncpus=2+1:ncpus=0:hps=1
```

This ensures that the hps resource is only counted once. You could also use this:

```
% qsub -l select=1:ncpus=8:hps=1
```

For two chunks of 4 CPUs, one on one machine and one on another, you would use:

```
% qsub -l select=2:ncpus=4 -l place=scatter
```

9.6.3.6 Useful Information

9.6.3.6.1 IBM Documentation

- IBM has documentation describing an InfiniBand cluster: “Clustering systems using InfiniBand hardware”, available from IBM at:

  `http://publib.boulder.ibm.com/infocenter/systems/scope/hw/index.jsp?topic=/iphau/referenceinfiniband.htm`

- IBM offers guidance on how to use the IB switch in US mode without LoadLeveler (PE v4.3.2 doc): “Configuring InfiniBand for User Space without LoadLeveler (PE for AIX only)”, available from IBM at:

  `http://publib.boulder.ibm.com/infocenter/clresctr/vxrx/index.jsp?topic=/com.ibm.cluster.pe432.install.doc/am101_configusib.html`

- IBM offers a programming API for communicating with and configuring the InfiniBand switch: “NRT API Programming Guide”.

9.6.3.6.2 Sources for Sample Code

When installing the `ppe.poe` fileset there are three directories containing sample code that may be of interest (from "How installing the POE fileset alters your system"):

- `/usr/lpp/ppe.poe/samples/swtbl`

  Directory containing sample code for running User Space POE jobs without LoadLeveler

- `/usr/lpp/ppe.poe/samples/ntbl`

  Directory containing sample code for running User Space jobs without LoadLeveler, using the network table API

- `/usr/lpp/ppe.poe/samples/nrt`

  Directory that contains the sample code for running User Space jobs on InfiniBand interconnects, without LoadLeveler, using the network resource table API.

9.6.3.7 Notes

Since PBS is tracking tasks started by poe, these tasks are counted towards a user's run limits. Running multiple poe jobs in the background will not work. Instead, run poe jobs one after the other or submit separate jobs. Otherwise HPS windows will be used by more than one task. The tracejob command will show any of various error messages.

For more information on using IBM's Parallel Operating Environment, see "IBM Parallel Environment for AIX 5L Hitchhiker's Guide".

9.6.3.8 Examples Using poe

Example 1: Using IP mode, run a single executable poe job with 4 ranks on hosts spread across the PBS-allocated nodes listed in $PBS_NODEFILE:

```
% cat $PBS_NODEFILE
host1
host2
host3
host4
% cat job.script
poe /path/mpiprog -euilib ip

% qsub -l select=4:ncpus=1 -lplace=scatter
    job.script
```

Example 2: Using US mode, run a single executable poe job with 4 ranks on hosts spread across the PBS-allocated nodes listed in $PBS_NODEFILE:

```
% cat $PBS_NODEFILE
host1
host2
host3
host4

% cat job.script
poe /path/mpiprog -euilib us

% qsub -l select=4:ncpus=1 -lplace=scatter
    job.script
```

Example 3: Using IP mode, run executables prog1 and prog2 with 2 ranks of prog1 on host1, 2 ranks of prog2 on host2 and 2 ranks of prog2 on host3.

```
% cat $PBS_NODEFILE
host1
host1
host2
host2
host3
host3
```

```
% cat job.script
echo prog1 > /tmp/poe.cmd
echo prog1 >> /tmp/poe.cmd
echo prog2 >> /tmp/poe.cmd
echo prog2 >> /tmp/poe.cmd
echo prog2 >> /tmp/poe.cmd
echo prog2 >> /tmp/poe.cmd
poe -cmdfile /tmp/poe.cmd -euilib ip
rm /tmp/poe.cmd
```

```
% qsub -l select=3:ncpus=2:mpiprocs=2 \
    -l place=scatter job.script
```

Example 4: Using US mode, run executables prog1 and prog2 with 2 ranks of prog1 on host1, 2 ranks of prog2 on host2 and 2 ranks of prog2 on host3.

```
% cat $PBS_NODEFILE
host1
host1
host2
host2
host3
host3
```

```
% cat job.script
echo prog1 > /tmp/poe.cmd
echo prog1 >> /tmp/poe.cmd
echo prog2 >> /tmp/poe.cmd
echo prog2 >> /tmp/poe.cmd
echo prog2 >> /tmp/poe.cmd
echo prog2 >> /tmp/poe.cmd
poe -cmdfile /tmp/poe.cmd -euilib us
rm /tmp/poe.cmd
```

```
% qsub -l select=3:ncpus=2:mpiprocs=2 \
    -l place=scatter job.script
```

9.6.4 PBS MPI Jobs on HP-UX and Linux

PBS is tightly integrated with the `mpirun` command on HP-UX so that resources can be tracked and processes managed. When running a PBS MPI job, you can use the same arguments to the `mpirun` command as you

would outside of PBS. The -h host and -l user options will be ignored, and the -np number option will be modified to fit the available resources.

9.6.5 PBS Jobs with MPICH-GM's mpirun Using rsh/ssh (mpirun.ch_gm)

PBS provides an interface to MPICH-GM's mpirun using rsh/ssh. If executed inside a PBS job, this lets PBS track all MPICH-GM processes started via rsh/ssh so that PBS can perform accounting and have complete job control. If executed outside of a PBS job, it behaves exactly as if standard mpirun.ch_gm had been used.

You use the same command as you would use outside of PBS, either "mpirun.ch_gm" or "mpirun".

9.6.5.1 Options

Inside a PBS job script, all of the options to the PBS interface are the same as mpirun.ch_gm except for the following:

-machinefile <file>

The file argument contents are ignored and replaced by the contents of the $PBS_NODEFILE.

-np

If not specified, the number of entries found in the $PBS_NODEFILE is used. The maximum number of ranks that can be launched is the number of entries in $PBS_NODEFILE.

-pg

The use of the -pg option, for having multiple executables on multiple hosts, is allowed but it is up to user to make sure only PBS hosts are specified in the process group file; MPI processes spawned on non-PBS hosts are not guaranteed to be under the control of PBS.

9.6.5.2 Examples

Example 1: Run a single-executable `MPICH-GM` job with 64 processes spread out across the PBS-allocated hosts listed in `$PBS_NODEFILE`:

PBS_NODEFILE:

```
pbs-host1
pbs-host2
pbs-host3
```

```
qsub -l select=3:ncpus=1
mpirun.ch_gm -np 64 /path/myprog.x 1200
^D
<job-id>
```

Example 2: Run an `MPICH-GM` job with multiple executables on multiple hosts listed in the process group file "`procgrp`":

```
qsub -l select=3:ncpus=1
echo  "host1 1 user1 /x/y/a.exe arg1 arg2" > procgrp
echo  "host2 1 user1 /x/x/b.exe arg1 arg2" >> procgrp
mpirun.ch_gm -pg procgrp /path/mypro.x
rm -f procgrp
^D
<job-id>
```

When the job runs, mpirun.ch_gm will give this warning message:

```
warning: "-pg" is allowed but it is up to user to make
   sure only PBS hosts are specified; MPI processes
   spawned are not guaranteed to be under the control
   of PBS.
```

The warning is issued because if any of the hosts listed in procgrp are not under the control of PBS, then the processes on those hosts will not be under the control of PBS.

9.6.6 PBS Jobs with MPICH-MX's mpirun Using rsh/ssh (mpirun.ch_mx)

PBS provides an interface to MPICH-MX's mpirun using rsh/ssh. If executed inside a PBS job, this allows for PBS to track all MPICH-MX processes started by rsh/ssh so that PBS can perform accounting and has complete job control. If executed outside of a PBS job, it behaves exactly as if standard mpirun.ch_mx had been used.

You use the same command as you would use outside of PBS, either "mpirun.ch_mx" or "mpirun".

9.6.6.1 Options

Inside a PBS job script, all of the options to the PBS interface are the same as mpirun.ch_mx except for the following:

-machinefile <file>
: The file argument contents are ignored and replaced by the contents of the $PBS_NODEFILE.

-np
: If not specified, the number of entries found in the $PBS_NODEFILE is used. The maximum number of ranks that can be launched is the number of entries in $PBS_NODEFILE.

-pg

The use of the -pg option, for having multiple executables on multiple hosts, is allowed but it is up to user to make sure only PBS hosts are specified in the process group file; MPI processes spawned on non-PBS hosts are not guaranteed to be under the control of PBS.

9.6.6.2 Examples

Example 1: Run a single-executable MPICH-MX job with 64 processes spread out across the PBS-allocated hosts listed in $PBS_NODEFILE:

PBS_NODEFILE:

```
pbs-host1
pbs-host2
pbs-host3
```

```
qsub -l select=3:ncpus=1
mpirun.ch_mx -np 64 /path/myprog.x 1200
^D
<job-id>
```

Example 2: Run an MPICH-MX job with multiple executables on multiple hosts listed in the process group file "procgrp":

```
qsub -l select=2:ncpus=1
echo  "pbs-host1 1 username /x/y/a.exe arg1 arg2" >
   procgrp
echo  "pbs-host2 1 username /x/x/b.exe arg1 arg2" >>
   procgrp
mpirun.ch_mx -pg procgrp /path/myprog.x
rm -f procgrp
^D
<job-id>
```

`mpirun.ch_mx` will give the warning message:

```
warning: "-pg" is allowed but it is up to user to make
   sure only PBS hosts are specified; MPI processes
   spawned are not guaranteed to be under PBS-control
```

The warning is issued because if any of the hosts listed in procgrp are not under the control of PBS, then the processes on those hosts will not be under the control of PBS.

9.6.7 PBS Jobs with MPICH-GM's mpirun Using MPD (mpirun.mpd)

PBS provides an interface to `MPICH-GM's mpirun` using MPD. If executed inside a PBS job, this allows for PBS to track all `MPICH-GM` processes started by the MPD daemons so that PBS can perform accounting have and complete job control. If executed outside of a PBS job, it behaves exactly as if standard `mpirun.mpd` with MPD had been used.

You use the same command as you would use outside of PBS, either "mpirun.mpd" or "mpirun". If the MPD daemons are not already running, the PBS interface will take care of starting them for you.

9.6.7.1 Options

Inside a PBS job script, all of the options to the PBS interface are the same as `mpirun.mpd` with MPD except for the following:

-m <file>
: The `file` argument contents are ignored and replaced by the contents of the `$PBS_NODEFILE`.

-np
: If not specified, the number of entries found in the `$PBS_NODEFILE` is used. The maximum number of ranks that can be launched is the number of entries in $PBS_NODEFILE

-pg

The use of the -pg option, for having multiple executables on multiple hosts, is allowed but it is up to user to make sure only PBS hosts are specified in the process group file; MPI processes spawned on non-PBS hosts are not guaranteed to be under the control of PBS.

9.6.7.2 MPD Startup and Shutdown

The script starts MPD daemons on each of the unique hosts listed in `$PBS_NODEFILE`, using either the `rsh` or `ssh` method based on the value of the environment variable `RSHCOMMAND`. The default is `rsh`. The script also takes care of shutting down the MPD daemons at the end of a run.

If the MPD daemons are not running, the PBS interface to `mpirun.mpd` will start GM's MPD daemons as this user on the allocated PBS hosts. The MPD daemons may have been started already by the administrator or by the user. MPD daemons are not started inside a PBS prologue script since it won't have the path of `mpirun.mpd` that the user executed (GM or MX), which would determine the path to the MPD binary.

9.6.7.3 Examples

Example 1: Run a single-executable `MPICH-GM` job with 64 processes spread out across the PBS-allocated hosts listed in `$PBS_NODEFILE`:

```
PBS_NODEFILE:
pbs-host1
pbs-host2
pbs-host3
qsub -l select=3:ncpus=1
[MPICH-GM-HOME]/bin/mpirun.mpd -np 64 /path/myprog.x
   1200
^D
<job-id>
```

If the GM MPD daemons are not running, the PBS interface to `mpirun.mpd` will start them as this user on the allocated PBS hosts.

The daemons may have been previously started by the administrator or the user.

Example 2: Run an MPICH-GM job with multiple executables on multiple hosts listed in the process group file "procgrp":

Job script:

```
qsub -l select=3:ncpus=1
echo  "host1 1 user1 /x/y/a.exe arg1 arg2" > procgrp
echo  "host2 1 user1 /x/x/b.exe arg1 arg2" >> procgrp

[MPICH-GM-HOME]/bin/mpirun.mpd -pg procgrp /path/
   mypro.x 1200
rm -f procgrp
^D
<job-id>
```

When the job runs, mpirun.mpd will give the warning message:

```
warning: "-pg" is allowed but it is up to user to make
   sure only PBS hosts are specified; MPI processes
   spawned are not guaranteed to be under PBS-control.
```

The warning is issued because if any of the hosts listed in procgrp are not under the control of PBS, then the processes on those hosts will not be under the control of PBS.

9.6.8 PBS Jobs with MPICH-MX's mpirun Using MPD (mpirun.mpd)

PBS provides an interface to MPICH-MX's mpirun using MPD. If executed inside a PBS job, this allows for PBS to track all MPICH-MX processes started by the MPD daemons so that PBS can perform accounting and have complete job control. If executed outside of a PBS job, it behaves exactly as if standard mpirun.ch_mx with MPD was used.

You use the same command as you would use outside of PBS, either "mpirun.mpd" or "mpirun". If the MPD daemons are not already running, the PBS interface will take care of starting them for you.

9.6.8.1 Options

Inside a PBS job script, all of the options to the PBS interface are the same as `mpirun.ch_gm` with MPD except for the following:

-m <file>

The `file` argument contents are ignored and replaced by the contents of the `$PBS_NODEFILE`.

-np

If not specified, the number of entries found in the `$PBS_NODEFILE` is used. The maximum number of ranks that can be launched is the number of entries in $PBS_NODEFILE.

-pg

The use of the `-pg` option, for having multiple executables on multiple hosts, is allowed but it is up to user to make sure only PBS hosts are specified in the process group file; MPI processes spawned on non-PBS hosts are not guaranteed to be under the control of PBS.

9.6.8.2 MPD Startup and Shutdown

The PBS `mpirun` interface starts MPD daemons on each of the unique hosts listed in `$PBS_NODEFILE`, using either the `rsh` or `ssh` method, based on value of environment variable `RSHCOMMAND`. The default is `rsh`. The interface also takes care of shutting down the MPD daemons at the end of a run.

If the MPD daemons are not running, the PBS interface to `mpirun.mpd` will start MX's MPD daemons as this user on the allocated PBS hosts. The MPD daemons may already have been started by the administrator or by the user. MPD daemons are not started inside a PBS prologue script since it won't have the path of `mpirun.mpd` that the user executed (GM or MX), which would determine the path to the MPD binary.

9.6.8.3 Examples

Example 1: Run a single-executable `MPICH-MX` job with 64 processes spread out across the PBS-allocated hosts listed in `$PBS_NODEFILE`:

PBS_NODEFILE:

```
pbs-host1
pbs-host2
pbs-host3
```

```
qsub -l select=3:ncpus=1
[MPICH-MX-HOME]/bin/mpirun.mpd -np 64 /path/myprog.x
   1200
^D
<job-id>
```

If the MPD daemons are not running, the PBS interface to `mpirun.mpd` will start GM's MPD daemons as this user on the allocated PBS hosts. The MPD daemons may be already started by the administrator or by the user.

Example 2: Run an `MPICH-MX` job with multiple executables on multiple hosts listed in the process group file “procgrp”:

```
qsub -l select=2:ncpus=1
echo  "pbs-host1 1 username /x/y/a.exe \
    arg1 arg2" > procgrp
echo  "pbs-host2 1 username /x/x/b.exe \
    arg1 arg2" >> procgrp
[MPICH-MX-HOME]/bin/mpirun.mpd -pg procgrp \
    /path/myprog.x 1200
rm -f procgrp
^D
<job-id>
```

`mpirun.mpd` will print a warning message:

warning: “-pg” is allowed but it is up to user to make sure only PBS hosts are specified; MPI processes spawned are not guaranteed to be under PBS-control

The warning is issued because if any of the hosts listed in `procgrp` are not under the control of PBS, then the processes on those hosts will not be under the control of PBS.

9.6.9 PBS Jobs with `MPICH2's mpirun`

PBS provides an interface to `MPICH2's mpirun`. If executed inside a PBS job, this allows for PBS to track all `MPICH2` processes so that PBS can perform accounting and have complete job control. If executed outside of a PBS job, it behaves exactly as if standard `MPICH2's mpirun` had been used.

You use the same “mpirun” command as you would use outside of PBS.

When submitting PBS jobs that invoke the pbsrun wrapper script for MPICH2's mpirun, be sure to explicitly specify the actual number of ranks or MPI tasks in the qsub select specification. Otherwise, jobs will fail to run with "too few entries in the machinefile".

For instance, specification of the following in 7.1:

```
#PBS -l
   select=1:ncpus=1:host=hostA+1:ncpus=2:host=hostB
mpirun -np 3 /tmp/mytask
```

would result in a 7.1 $PBS_NODEFILE listing:

```
hostA
hostB
hostB
```

but in 8.0 or later it would be:

```
hostA
hostB
```

which would conflict with the "-np 3" specification in mpirun as only 2 MPD daemons will be started.

The correct way now is to specify either a) or b) as follows:

a. `#PBS -l`
`select=1:ncpus=1:host=`hostA+2:ncpus=1:host=hostB

b. #PBS -l
select=1:ncpus=1:host=hostA+1:ncpus=2:host=hostB:mpiprocs=2

which will cause $PBS_NODEFILE to list:

```
hostA
hostB
hostB
```

and an "mpirun -np 3" will then be consistent.

9.6.9.1 Options

If executed inside a PBS job script, all of the options to the PBS interface are the same as `MPICH2's mpirun` except for the following:

-host, -ghost
: For specifying the execution host to run on. Ignored.

-machinefile <file>
: The file argument contents are ignored and replaced by the contents of the `$PBS_NODEFILE.`

-localonly <x>
: For specifying the <x> number of processes to run locally. Not supported. The user is advised instead to use the equivalent arguments:

  `"-np <x> -localonly".`

-np
: If the user does not specify a -np option, then no default value is provided by the PBS wrapper scripts. It is up to the local mpirun to decide what the reasonable default value should be, which is usually 1. The maximum number of ranks that can be launched is the number of entries in $PBS_NODEFILE.

9.6.9.2 MPD Startup and Shutdown

The interface ensures that the MPD daemons are started on each of the hosts listed in the `$PBS_NODEFILE`. It also ensures that the MPD daemons are shut down at the end of MPI job execution.

9.6.9.3 Examples

Example 1: Run a single-executable MPICH2 job with 6 processes spread out across the PBS-allocated hosts listed in $PBS_NODEFILE:

PBS_NODEFILE:

```
pbs-host1
pbs-host2
pbs-host3
pbs-host1
pbs-host2
pbs-host3
```

Job.script:

```
# mpirun runs 6 processes mapped to each host
# listed in $PBS_NODEFILE
mpirun -np 6 /path/myprog.x 1200
```

Run job script:

```
qsub -l select=3:ncpus=2 job.script
```

```
<job-id>
```

Example 2: Run an MPICH2 job with multiple executables on multiple hosts using $PBS_NODEFILE and mpiexec arguments in mpirun:

PBS_NODEFILE:

```
hostA
hostA
hostB
hostB
hostC
hostC
```

Job script:

```
#PBS -l select=3:ncpus=2
mpirun -np 2 /tmp/mpitest1 : \
   -np 2 /tmp/mpitest2 : \
   -np 2 /tmp/mpitest3
```

Run job:

```
qsub job.script
```

Example 3: Run an MPICH2 job with multiple executables on multiple hosts using `mpirun -configfile` option and `$PBS_NODEFILE`:

PBS_NODEFILE:

```
hostA
hostA
hostB
hostB
hostC
hostC
```

Job script:

```
#PBS -l select=3:ncpus=2
echo "-np 2 /tmp/mpitest1" > my_config_file
echo "-np 2 /tmp/mpitest2" >> my_config_file
echo "-np 2 /tmp/mpitest3" >> my_config_file
mpirun -configfile my_config_file
rm -f my_config_file
```

Run job:

```
qsub job.script
```

9.6.10 PBS Jobs with Intel MPI's `mpirun`

PBS provides an interface to Intel MPI's `mpirun`. If executed inside a PBS job, this allows for PBS to track all Intel MPI processes so that PBS can perform accounting and have complete job control. If executed outside of a PBS job, it behaves exactly as if standard Intel MPI's `mpirun` was used.

You use the same "mpirun" command as you would use outside of PBS.

When submitting PBS jobs that invoke the pbsrun wrapper script for Intel MPI, be sure to explicitly specify the actual number of ranks or MPI tasks in the qsub select specification. Otherwise, jobs will fail to run with "too few entries in the machinefile".

For instance, specification of the following in 7.1:

```
#PBS -l
   select=1:ncpus=1:host=hostA+1:ncpus=2:host=hostB
mpirun -np 3 /tmp/mytask
```

would result in a 7.1 $PBS_NODEFILE listing:

```
hostA
hostB
hostB
```

but in 8.0 or later it would be:

```
hostA
hostB
```

which would conflict with the "-np 3" specification in mpirun as only 2 MPD daemons will be started.

The correct way now is to specify either a) or b) as follows:

a. #PBS -l select=1:ncpus=1:host=hostA+2:ncpus=1:host=hostB

b. #PBS -l
select=1:ncpus=1:host=hostA+1:ncpus=2:host=hostB:mpiprocs=2

which will cause $PBS_NODEFILE to list:

```
hostA
hostB
hostB
```

and an "mpirun -np 3" will then be consistent.

9.6.10.1 Options

If executed inside a PBS job script, all of the options to the PBS interface are the same as for Intel MPI's `mpirun` except for the following:

-host, -ghost
: For specifying the execution host to run on. Ignored.

-machinefile <file>
: The file argument contents are ignored and replaced by the contents of the `$PBS_NODEFILE`.

mpdboot option --totalnum=*
: Ignored and replaced by the number of unique entries in `$PBS_NODEFILE`.

mpdboot option --file=*
: Ignored and replaced by the name of `$PBS_NODEFILE.` The argument to this option is replaced by `$PBS_NODEFILE`.

  Argument to `mpdboot` option `-f <mpd_hosts_file>` replaced by `$PBS_NODEFILE`.

-s
: If the PBS interface to Intel MPI's `mpirun` is called inside a PBS job, Intel MPI's `mpirun -s` argument to `mpdboot` is not supported as this closely matches the `mpirun` option "`-s <spec>`". The user can simply run a separate `mpdboot -s` before calling `mpirun`. A warning message is issued by the PBS interface upon encountering a `-s` option telling users of the supported form.

-np
: If the user does not specify a `-np` option, then no default value is provided by the PBS interface. It is up to the local `mpirun` to decide what the reasonable default value should

be, which is usually 1. The maximum number of ranks that can be launched is the number of entries in $PBS_NODEFILE.

9.6.10.2 MPD Startup and Shutdown

Intel MPI's `mpirun` takes care of starting/stopping the MPD daemons. The PBS interface to Intel MPI's `mpirun` always passes the arguments `-totalnum=<number of mpds to start>` and `-file=<mpd_hosts_file>` to the actual `mpirun`, taking its input from unique entries in `$PBS_NODEFILE`.

9.6.10.3 Examples

Example 1: Run a single-executable Intel MPI job with 6 processes spread out across the PBS-allocated hosts listed in `$PBS_NODEFILE`:

PBS_NODEFILE:

```
pbs-host1
pbs-host2
pbs-host3
pbs-host1
pbs-host2
pbs-host3
```

Job script:

```
# mpirun takes care of starting the MPD
# daemons on unique hosts listed in
# $PBS_NODEFILE, and also runs 6 processes
# mapped to each host listed in
# $PBS_NODEFILE; mpirun takes care of
# shutting down MPDs.
mpirun /path/myprog.x 1200
```

Run job script:

```
qsub -l select=3:ncpus=2 job.script
<job-id>
```

Example 2: Run an Intel MPI job with multiple executables on multiple hosts using `$PBS_NODEFILE` and `mpiexec` arguments to `mpirun`:

$PBS_NODEFILE

```
hostA
hostA
hostB
hostB
hostC
hostC
```

Job script:

```
# mpirun runs MPD daemons on hosts listed
# in $PBS_NODEFILE
# mpirun runs 2 instances of mpitest1
# on hostA; 2 instances of mpitest2 on
# hostB; 2 instances of mpitest3 on
# hostC.
# mpirun takes care of shutting down the
# MPDs at the end of MPI job run.
mpirun -np 2 /tmp/mpitest1 : -np 2 /tmp/mpitest2 : -np
   2 /tmp/mpitest3
```

Run job script:

```
qsub -l select=3:ncpus=2 job.script
 <job-id>
```

Example 3: Run an Intel MPI job with multiple executables on multiple hosts via the -configfile option and $PBS_NODEFILE:

$PBS_NODEFILE:

```
hostA
hostA
hostB
hostB
hostC
hostC
```

Job script:

```
echo "-np 2 /tmp/mpitest1" >> my_config_file
echo "-np 2 /tmp/mpitest2" >> my_config_file
echo "-np 2 /tmp/mpitest3" >> my_config_file

# mpirun takes care of starting the MPD daemons
# config file says run 2 instances of mpitest1
# on hostA; 2 instances of mpitest2 on
# hostB; 2 instances of mpitest3 on
# hostC.
# mpirun takes care of shutting down the MPD
# daemons.
mpirun -configfile my_config_file

# cleanup
rm -f my_config_file
```

Run job script:

```
qsub -l select=3:ncpus=2 job.script
<job-id>
```

9.6.11 PBS Jobs with `MVAPICH1's mpirun`

PBS provides an interface to `MVAPICH1's mpirun`. MVAPICH1 allows use of InfiniBand. If executed inside a PBS job, this allows for PBS to track all `MVAPICH1` processes so that PBS can perform accounting and have complete job control. If executed outside of a PBS job, it behaves exactly as if standard `MVAPICH1 mpirun` had been used.

You use the same "mpirun" command as you would use outside of PBS.

9.6.11.1 Options

If executed inside a PBS job script, all of the options to the PBS interface are the same as `MVAPICH1's mpirun` except for the following:

-map
: The map option is ignored.

`-machinefile <file>`
: The machinefile option is ignored.

`-exclude`
: The exclude option is ignored.

-np
: If the user does not specify a `-np` option, then PBS uses the number of entries found in the $PBS_NODEFILE. The maximum number of ranks that can be launched is the number of entries in $PBS_NODEFILE.

9.6.11.2 Examples

Example 1: Run a single-executable `MVAPICH1` job with 6 ranks spread out across the PBS-allocated hosts listed in `$PBS_NODEFILE`:

PBS_NODEFILE:

```
pbs-host1
pbs-host1
pbs-host2
pbs-host2
pbs-host3
pbs-host3
```

Job.script:

```
# mpirun runs 6 processes mapped to each host listed
# in $PBS_NODEFILE
mpirun -np 6 /path/myprog
```

Run job script:

```
qsub -l select=3:ncpus=2:mpiprocs=2 job.script
<job-id>
```

9.6.12 PBS Jobs with `MVAPICH2's mpiexec`

PBS provides an interface to `MVAPICH2's mpiexec`. MVAPICH2 allows the use of InfiniBand. If executed inside a PBS job, this allows for PBS to track all `MVAPICH2` processes so that PBS can perform accounting and have complete job control. If executed outside of a PBS job, it behaves exactly as if standard `MVAPICH2's mpiexec` had been used.

You use the same “mpiexec” command as you would use outside of PBS.

The maximum number of ranks that can be launched is the number of entries in $PBS_NODEFILE.

9.6.12.1 Options

If executed inside a PBS job script, all of the options to the PBS interface are the same as `MVAPICH2's mpiexec` except for the following:

-host
: The host option is ignored.

-machinefile <file>
: The file option is ignored.

-mpdboot
: If mpdboot is not called before mpiexec, it is called automatically before `mpiexec` runs so that an MPD daemon is started on each host assigned by PBS.

9.6.12.2 MPD Startup and Shutdown

The interface ensures that the MPD daemons are started on each of the hosts listed in the `$PBS_NODEFILE`. It also ensures that the MPD daemons are shut down at the end of MPI job execution.

9.6.12.3 Examples

Example 1: Run a single-executable `MVAPICH2` job with 6 ranks on hosts listed in `$PBS_NODEFILE`:

PBS_NODEFILE:

```
pbs-host1
pbs-host2
pbs-host3
```

Job.script:

```
mpiexec -np 6 /path/mpiprog
```

Run job script:

```
qsub -l select=3:ncpus=2 job.script
<job-id>
```

Example 2: Launch an MVAPICH2 MPI job with multiple executables on multiple hosts listed in the default file "mpd.hosts". Here, run executables prog1 and prog2 with 2 ranks of prog1 on host1, 2 ranks of prog2 on host2 and 2 ranks of prog2 on host3 all specified on the command line.

PBS_NODEFILE:

```
pbs-host1
pbs-host2
pbs-host3
```

Job.script:

```
mpiexec -n 2 prog1 : -n 2  prog2 : -n 2  prog2
```

Run job script:

```
qsub -l select=3:ncpus=2 job.script
<job-id>
```

Example 3: Launch an MVAPICH2 MPI job with multiple executables on multiple hosts listed in the default file "mpd.hosts". Run executables prog1 and prog2 with 2 ranks of prog1 on host1, 2 ranks of prog2 on host2 and 2 ranks of prog2 on host3 all specified using the -configfile option.

PBS_NODEFILE:

```
pbs-host1
pbs-host2
pbs-host3
```

Job.script:

```
echo "-n 2 -host host1 prog1" > /tmp/jobconf
echo "-n 2 -host host2 prog2" >> /tmp/jobconf
echo "-n 2 -host host3 prog2" >> /tmp/jobconf
mpiexec -configfile /tmp/jobconf
rm /tmp/jobconf
```

Run job script:

```
qsub -l select=3:ncpus=2 job.script
<job-id>
```

9.6.13 PBS Jobs with HP MPI

In order to override the default rsh, set PBS_RSHCOMMAND in your job script:

```
export PBS_RSHCOMMAND=<rsh_cmd>
```

9.7 MPI Jobs on the Altix

9.7.1 Jobs on an Altix Running ProPack 6/7 or Performance Suite 1

PBS has its own mpiexec for the Altix running ProPack 6 or 7 or Performance Suite 1. The PBS mpiexec has the standard mpiexec interface. The PBS mpiexec does require proper configuration of the Altix. See your administrator to find out whether your system is configured for the PBS mpiexec.

You can launch an MPI job on a single Altix, or across multiple Altixes. PBS will manage and track the processes. You can use CSA, if it is configured, to collect accounting information on your jobs. PBS will run the MPI tasks in the cpusets it manages.

You can run MPI jobs in the placement sets chosen by PBS. When a job is finished, PBS will clean up after it.

For MPI jobs across multiple Altixes, PBS will manage the multihost jobs. For example, if you have two Altixes named Alt1 and Alt2, and want to run two applications called mympi1 and mympi2 on them, you can put this in your job script:

```
mpiexec -host Alt1 -n 4 mympi1 : -host Alt2 -n 8 mympi2
```

You can specify the name of the array to use via the PBS_MPI_SGIARRAY environment variable.

To verify how many CPUs are included in a cpuset created by PBS, use:

```
> $ cpuset -d <set name> | egrep cpus
```

This will work either from within a job or not.

The alt_id returned by MOM has the form `cpuset=<name>`. <name> is the name of the cpuset, which is the $PBS_JOBID.

Jobs will share cpusets if the jobs request sharing and the cpusets' sharing attribute is not set to force_excl. Jobs can share the memory on a nodeboard if they have a CPU from that nodeboard. To fit as many small jobs as possible onto vnodes that already have shared jobs on them, request sharing in the job resource requests.

PBS will try to put a job that will fit in a single nodeboard on just one nodeboard. However, if the only CPUs available are on separate nodeboards, and those vnodes are not allocated exclusively to existing jobs, and the job can share a vnode, then the job will be run on the separate nodeboards.

If a job is suspended, its processes will be moved to the global cpuset. When the job is restarted, they are restored.

9.7.2 Caveats

In order to use MPI within a PBS job with Performance Suite 1, you may need to add the following in your job script before you call MPI:

```
module load mpt
```

9.8 PVM Jobs with PBS

On a typical system, to execute a Parallel Virtual Machine (PVM) program you can use the `pvmexec` command. The pvmexec command expects a "hostfile" argument for the list of hosts on which to spawn the parallel job.

For example, here is a sample PBS script for a PVM job:

```
#PBS -N pvmjob
#
pvmexec a.out -inputfile data_in
```

To start the PVM daemons on the hosts listed in $PBS_NODEFILE, start the PVM console on the first host in the list, and print the hosts to the standard output file named "jobname.o<PBS jobID>, use "echo conf | pvm $PBS_NODEFILE". To quit the PVM console but leave the PVM daemons running, use "quit". To stop the PVM daemons, restart the PVM console, and quit, use "echo halt | pvm".

To submit a PVM job to PBS, use

```
qsub  your_pvm_job
```

Here is an example script for your_pvm_job:

```
#PBS -N pvmjob
#PBS -V
cd $PBS_O_WORKDIR
echo conf   | pvm $PBS_NODEFILE
echo quit | pvm
./my_pvm_program
echo halt | pvm
```

9.9 Checkpointing SGI MPI Jobs

9.9.1 Jobs on an Altix

Jobs are suspended on the Altix using the PBS suspend feature. Jobs are checkpointed on the Altix using application-level checkpointing. There is no OS-level checkpoint. Suspended or checkpointed jobs will resume on the original nodeboards.

Chapter 10

HPC Basic Profile Jobs

PBS Professional can schedule and manage jobs on one or more HPC Basic Profile compliant servers using the Grid Forum OGSA HPC Basic Profile web services standard. You can submit a generic job to PBS, so that PBS can run it on an HPC Basic Profile Server. This chapter describes how to use PBS for HPC Basic Profile jobs.

10.1 Definitions

HPC Basic Profile (HPCBP)
: Proposed standard web services specification for basic job execution capabilities defined by the OGSA High Performance Computing Profile Working Group

HPC Basic Profile Server
: Service that executes jobs from any HPC Basic Profile compliant client

HPCBP MOM

MOM that sends jobs for execution to an HPC Basic Profile Server. This MOM is a client-side implementation of the HPC Basic Profile Specification, and acts as a proxy for and interface to an HPC Basic Profile compliant server.

HPC Basic Profile Job, HPCBP Job

Generic job that can run either on vnodes managed by PBS or on nodes managed by HPC Basic Profile Server.

Job Submission Description Language (JSDL)

Language for describing the resource requirements of jobs

10.2 How HPC Basic Profile Jobs Work

10.2.1 Introduction

PBS automatically schedules jobs on vnodes managed by PBS Professional or on nodes managed by an HPC Basic Profile Server, without the need for you to specify destination-specific parameters. Whether the jobs run on PBS Professional or on an HPC Basic Profile Server is based only on site policies and resource availability.

You can use the `qstat` command for status reporting and the `qdel` command to cancel a job, regardless of where the job runs.

Jobs eligible to run on the HPCBP Server must specify only a single executable and its arguments, and must do so via the `qsub` command line. The job specification must be valid for both PBS and the HPCBP Server. A job that is eligible to run on the HPCBP Server is called an *HPCBP job* in this document.

10.2.2 Assigning Nodes and Resources to Jobs

The HPCBP MOM does not control the resources assigned from each node for a job. The HPC Basic Profile Server assigns resources to the job according to its scheduling policy.

If you specify HPCBP hosts as part of the job's select statement, the list of of HPCBP hosts is passed to the HPCBP Server.

10.3 Environmental Requirements for HPCBP

10.3.1 User Account at HPCBP Server

You must be able to run commands at the HPCBP Server. You must have an account in the Domain Controller at the HPCBP Server.

10.3.2 HPCBP Submission Client Architecture

You can submit HPCBP jobs only from submission hosts that have the correct architecture. These are all supported Linux platforms on x86 and x86_64.

10.3.3 Password Requirement For Job Submission

The HPC Basic Profile Server requires a password and a username to perform operations such as job submission, status, termination etc. The PBS Server must pass credential information to the HPCBP MOM at the time of job submission.

Before submitting an HPCBP job, you must run the `pbs_password` command to store your password at the PBS server. When you submit an HPCBP job, you must supply a password. This is done in one of two ways:

- The administrator sets the single_signon_password_enable server attribute to True
- You use the '-Wpwd' option to the `qsub` command to pass credential information to the PBS Server

10.3.4 Location of Executable

The executable that your job runs must be available at the HPC Server. The following table lists how the path to the executable can be specified:

Table 10-1: Executable Path Specification

Path Specification	Location of Executable
You can specify an absolute path to the executable	Anywhere available to the HPCBP Server
You can specify a path relative to your home directory on the HPC Server	A path relative to your home directory on the HPC Server
You can specify just the name of the executable	The executable is in your PATH or in your default working directory

10.4 Submitting HPC Basic Profile Jobs

As with PBS jobs, you do not need to specify destination-specific parameters.

10.4.1 Restrictions on Submitting Jobs for Execution at HPCBP Server

10.4.1.1 Specifying Executable for Job

The job must specify exactly one executable and its arguments. This must be done on the `qsub` command line.

10.4.1.2 HPCBP Jobs Run on One HPCBP Server

The job must not be split across more than one HPCBP Server:

- It cannot be split across two or more HPCBP Servers
- It cannot be split across an HPCBP Server and another node

10.4.1.3 Number of CPUs and mpiprocs

For each chunk, the aggregate number of requested ncpus must match the aggregate number of requested mpiprocs. The default value per chunk for both ncpus and mpiprocs is 1. If you request 1 CPU per chunk, you do not have to specify the mpiprocs. If the requested values for ncpus and mpiprocs are different, an error message is logged to the HPCBP MOM log file and the job is rejected. So for example if you request

```
qsub -l select=3:ncpus=2:mem=8gb
```

the job is rejected because no mpiprocs were requested.

10.4.1.4 Number of ompthreads

For a job with more than one chunk that requests ompthreads, each chunk must request the same value for ompthreads. Otherwise, an error message is logged to the HPCBP MOM log file and the job is rejected.

10.4.1.5 Restrictions on Requesting arch Resource

Requesting a value for arch in an HPCBP job means requesting a node or nodes with that architecture from among the nodes controlled by the HPCBP Server. It is not necessary for a job to request a value for arch. An HPCBP job can request any arch value that can be satisfied by the HPCBP Server.

10.4.2 Using the qsub Command for HPCBP Jobs

Job submission for non-HPCBP jobs is unchanged. However, when you submit an HPCBP job, you must do the following:

- Specify only one executable and its arguments
- Specify executable and arguments in the qsub command line

10.4.2.1 qsub Syntax for HPCBP Jobs

```
qsub [-a date_time] [-A account_string] [-c interval]
   [-C directive_prefix] [-e path] [-h ] [-I] [-j
   oe|eo] [-J X-Y[:Z]] [-k o|e|oe] [-l resource_list]
   [-m mail_options] [-M user_list] [-N jobname] [-o
   path] [-p priority] [-q queue] [-r y|n] [-S path] [-
   u user_list] [-W otherattributes=value...] [-v
   variable_list] [-V ] [-z] -- cmd [arg1...]
```

or

```
qsub --version
```

where cmd is the executable, and arg1 is the first argument in the list.

10.4.2.2 qsub Options for HPCBP Jobs

The options to the qsub command set the attributes for the job. The following table shows a list of PBS job attributes and their behavior for HPCBP jobs.

Table 10-2: Behavior of Job Attributes for HPCBP Jobs

PBS Job attribute	Behavior
interactive	Job is rejected with transient error
Resource List	See section 10.4.3 “Requesting Resources” on page 280
Output path	Standard output is staged out to specified location

Table 10-2: Behavior of Job Attributes for HPCBP Jobs

PBS Job attribute	Behavior
Error_path	Standard error is staged out to specified location
no_stdio_sockets	Unsupported
Shell_Path_List	Unsupported
Variable_List	User's environment is passed to HPCBP Server
alt_id	Set to job ID returned by HPC Server
exec_host	Same as standard. Set to list of hosts, with number of CPUs for each
exec_vnode	Same as standard. Set to list of vnodes, with number of CPUs and amount of memory
job_state	See section 10.5.1.1 "Job Status Reporting" on page 282
resources_used	Set to cputime used and amount of memory requested
session_id	Returns process ID of process started by the HPCBP MOM for job management, not of HPCBP job itself
stime	Reported start time of job; may be inexact
substate	The job substate may not be same in HPC Basic Profile Server and PBS
group_list	Unsupported
stagein	Specified files are staged in
stageout	Specified files are staged out
umask	Unsupported

10.4.3 Requesting Resources

The following table shows the behavior for of PBS resources HPCBP jobs:

Table 10-3: PBS Resources and Their Behavior for HPCBP Jobs

PBS Resource	Behavior
arch	Same as standard.
cput	Amount of disk space for job
file	Same as standard
host	Same as standard
mem	Same as standard
mpiprocs	Number of CPUs to be allocated to job
mppwidth	Unsupported
mppdepth	Unsupported
mppnppn	Unsupported
mppnodes	Unsupported
mpplabels	Unsupported
mppmem	Unsupported
mpphost	Unsupported
mpparch	Unsupported
ncpus	Same as standard
nice	Unsupported
nodect	Unsupported
ompthreads	Must specify equal number of ompthreads in all chunks of multi-chunk job
pcput	Same as standard
pmem	Same as standard

Table 10-3: PBS Resources and Their Behavior for HPCBP Jobs

PBS Resource	Behavior
pvmem	Same as standard
software	Unsupported
vmem	Same as standard
vnode	Same as standard
walltime	Supported
cpupercent	Unsupported
custom resources	Unsupported

10.4.4 Specifying Job Destination

If necessary, you can specify where your job should run. You can specify on which nodes you want to run your job by specifying a host name:

```
-lselect=host=<host name>
```

If your application can run only on Windows, then you should request PBS to run the job only on Windows HPC Server nodes by specifying the architecture:

```
-lselect=arch=<arch value returned from HPCBP MOM>
```

Similarly, if you want to run your application on Linux, then you need to specify that architecture:

```
-lselect=arch=linux
```

If you don't specify a value for the arch resource at the time of job submission, PBS will select vnodes based on availability and run your application there.

10.5 Managing HPCBP Jobs

10.5.1 Monitoring HPCBP Jobs

You can use `qstat -f <job ID>` to see a listing of your job's executable and its argument list.

For example, if your job request was:

```
qsub -- ping -n 100 127.0.0.1
```

The output of `qstat -f <job ID>` will be:

```
executable = <jsdl-hpcpa:Executable>ping</jsdl-
   hpcpa:Executable>
argument_list = <jsdl-hpcpa:Argument>-n</jsdl-
   hpcpa:Argument>  <jsdl-hpcpa:Argument>100</jsdl-
   hpcpa:Argument>  <jsdl-hpcpa:Argument>127.0.0.1</
   jsdl-hpcpa:Argument>
```

10.5.1.1 Job Status Reporting

PBS provides status reporting for HPC Basic Profile jobs via the `qstat` command. The HPCBP MOM contacts the HPC Basic Profile Server and returns status information to the PBS Server. The only information available is via the HPC Basic Profile.

The job states returned from HPC Basic Profile Server can be one of the following:

- Pending
- Running
- Failed
- Finished
- Terminated

However, the only states that are reported by `qstat` are

- Running
- Exiting

The HPCBP Server reports that the job is in *Running* state whether the job is waiting to run or is running.

Once a job transitions to any of the states *Terminated*, *Failed* or *Finished*, the HPCBP MOM will no longer query for the status of that job.

A job whose status is *Running* can become *Terminated*, *Failed*, or *Finished*, or *Exiting*.

10.5.1.2 Deleting jobs running at HPC Basic Profile Server

You can delete your jobs via the `qdel` command:

```
qdel <job ID>
```

10.6 Errors, Logging and Troubleshooting

10.6.1 Job Submission Password Problems

If you specify the wrong password, or the password is different from the one at the HPC Basic Profile Server:

- The HPCBP MOM rejects the job and the PBS Server sets the job's comment
- The PBS Server logs a message in the server log
- The PBS Server changes the state of the job to *Hold* and the substate to *waiting on dependency* and keeps it in the queue

10.6.2 Job Format Problems

If you submit only a job script, without any executable and argument list, and PBS attempts to run the job on the HPCBP Server, the HPCBP MOM will log a message and return an error.

If you submit a job requesting non-HPCBP vnodes and HPCBP nodes, or requesting nodes from two different HPCBP Servers:

- The job is rejected
- The HPCBP MOM logs an error message

10.6.3 Password-related Job Deletion Issues

If any problem, such as bad user credentials, occurs during an attempt to delete a job:

- The `qdel` command displays an error message
- The PBS server writes the error message to the server log
- The HPCBP MOM logs an error message

10.6.4 Error Log Messages at Job Submission, Querying, and Deletion

The HPCBP MOM logs a warning message in the MOM log file whenever it gets any error or warning at the time of:

- Job submission
- Contacting the HPC Basic Profile Server to find job status
- Job deletion

The HPCBP MOM logs job errors in the file `<PBS job ID>.log`. The HPCBP MOM stages this file out to the location specified for `stdout` and `stderr` files.

The HPCBP MOM generates log messages depending on their event type and event class. You can use the `tracejob` command to see these log messages.

The following table shows the warning and error messages logged by the HPCBP MOM and the PBS Server:

Table 10-4: Warning and Error Messages Logged by HPCBP MOM

Error Condition	Logged by	Message
Password-related issues		
Bad user credential at the time of `qdel`	HPCBP MOM, PBS Server	<username>: unable to terminate the job with user's credentials
Cannot determine job state when finding status of jobs running at HPC Basic Profile Server	HPCBP MOM	<pbsnobody>: unable to determine the state of the job
Conversion of PBS job request to JSDL		
Problem with parsing job request	HPCBP MOM	unable to parse the job request
Job request contains a script	HPCBP MOM	can't submit job to HPC Basic Profile Server, HPCBP MOM doesn't accept job script
JSDL script file problem	HPCBP MOM	unable to create JSDL document
gSOAP-related problems		
cannot create SSL-based channel	HPCBP MOM	unable to create ssl-based channel to connect to the Web Service endpoint
Username token problem	HPCBP MOM	unable to add username/password to soap message

Table 10-4: Warning and Error Messages Logged by HPCBP MOM

Error Condition	Logged by	Message
Cannot initialize gSOAP runtime environment	HPCBP MOM	unable to initialize gsoap runtime environment
Problems encountered during job submission		
Cannot add SOAP Header	HPCBP MOM	unable to add soap header to the 'create activity' request message
Bad JSDL script file	HPCBP MOM	unable to open JSDL document
Problem with JSDL attribute	HPCBP MOM	error in reading contents of the JSDL document
Problem with HPCBP Server connection	HPCBP MOM	unable to submit job to the hpcbp web service endpoint
Problem with user's password	HPCBP MOM & PBS Server	unable to submit job with user's credential
Problem reading SOAP response	HPCBP MOM	unable to read HPCBP job identifier from create activity response
Problems encountered when deleting job		
Cannot add SOAP Header	HPCBP MOM	unable to add SOAP Header to the 'terminate activities' request message
Problem reading HPCBP job ID	HPCBP MOM	unable to read HPCBP job identifier
Bad HPC Basic Profile Server connection	HPCBP MOM	unable to connect to the HPCBP web service endpoint

Table 10-4: Warning and Error Messages Logged by HPCBP MOM

Error Condition	Logged by	Message
Problem with user's password	HPCBP MOM, PBS Server	unable to terminate job with user's credentials
Received malformed response from HPCBP Server	HPCBP MOM	unable to parse the response received for job deletion request from HPCBP Server
Problems encountered when finding status of job		
Cannot add SOAP Header	HPCBP MOM	unable to add SOAP Header to the 'get activity statuses' request message
Problem reading HPCBP JOB ID	HPCBP MOM	unable to read HPCBP job identifier
Bad HPC Basic Profile Server connection	HPCBP MOM	unable to connect to the HPCBP web service endpoint
Received malformed response from HPCBP Server	HPCBP MOM	unable to parse the job status response received from HPCBP Server
Problems encountered when finding node status		
Cannot add SOAP Header	HPCBP MOM	unable to add SOAP Header to the 'get factory attributes document' request message
Problem with reading the node status information	HPCBP MOM	unable to parse node status information received from the HPC Basic Profile Server
HPC Basic Profile Server Connection	HPCBP MOM	unable to connect to the HPCBP web service endpoint
mpiprocs-related error		

Table 10-4: Warning and Error Messages Logged by HPCBP MOM

Error Condition	Logged by	Message
unequal ncpus & mpiprocs	HPCBP MOM	can't submit job to the HPC Basic Profile Server; total number of ncpus and mpiprocs requested are not equal
ompthreads error		
ompthreads are not equal across chunks	HPCBP MOM	can't submit job to the HPC Basic Profile Server; number of 'ompthreads' are not equal in multi-chunk job request
Generic Problems		
No reply from HPCBP Server	HPCBP MOM	unable to receive response from hpcbp web service endpoint
OpenSSL library issues		
Cannot find OpenSSL libraries on system	HPCBP MOM	unable to find openssl libraries on the system.

10.6.5 Job State Transition Log Messages

See the following table for a list of the job transitions in the HPCBP Server and the associated actions by the HPCBP MOM:

Table 10-5: Job Transitions in HPCBP Server and Associated Actions by HPCBP MOM

Job Transitions in HPC Basic Profile Server		Message Logged By HPCBP MOM
Start State	End State	
Pending	*Running*	"job transitioned from pending to running

Table 10-5: Job Transitions in HPCBP Server and Associated Actions by HPCBP MOM

Job Transitions in HPC Basic Profile Server		Message Logged By HPCBP MOM
Start State	End State	
Pending	*Terminated*	"job transitioned from pending to terminated"
Running	*Terminated*	"job transitioned from running to terminated"
Running	*Failed*	"job transitioned from running to failed"
Running	*Finished*	"job completed successfully"
Pending	*Finished*	"job transitioned from pending to finished"
Pending	*Failed*	"job transitioned from pending to failed"
(none)	*Failed*	"job first appeared in "Failed" state"

Whenever a job is submitted to the HPC Basic Profile Server, the HPCBP MOM logs the following message:

```
job submitted to HPCBP Server as jobid <hpcbp-jobid> in
   state <state>
```

10.7 Advice and Caveats

10.7.1 Differences Between PBS and HPCBP

- The stime attribute in the PBS accounting logs may not represent the exact start time for an HPCBP job.
- The HPCBP MOM does not use the `pbs_rcp` command for staging operations, regardless of whether the PBS_SCP environment variable has been set in the configuration file.

10.7.2 PBS Features Not Supported With HPCBP

- Peer Scheduling
- Job operations:
  - Suspend/resume
  - Checkpoint

10.7.2.1 Unsupported Commands

If the user or administrator runs the `pbsdsh` command for a job running on the HPCBP Server, the HPCBP MOM logs an error message to the MOM file and rejects the job.

The following commands and their API equivalents are not supported for jobs that end up running on the HPCBP Server:

- `qalter`
- `qsig`
- `qmsg`
- `pbsdsh`
- `pbs-report`
- `printjob`
- `pbs_rcp`
- `tracejob`
- `pbs_rsub`
- `pbs_rstat`
- `pbs_rdel`
- `qhold`
- `qrls`
- `qrerun`

10.8 See Also

For a description of how job attributes are translated into JSDL, see the PBS Professional External Reference Specification.

10.8.1 References

1. OGSA High Performance Computing Profile Working Group (OGSA-HPCP-WG) of the Open Grid Forum

   `https://forge.gridforum.org/sf/projects/ogsa-hpcp-wg`

   The HPC Basic Profile specification is GFD.114:

   `http://www.ogf.org/documents/GFD.114.pdf.`

2. OGSA High Performance Computing Profile Working Group (OGSA-HPCP-WG) of the Open Grid Forum

   `https://forge.gridforum.org/sf/projects/ogsa-hpcp-wg`

   The HPC File Staging Profile Version 1.0:

   `http://forge.ogf.org/sf/go/doc15024?nav=1`

3. OGSA Job Submission Description Language Working Group (JSDL - WG) of the Open Grid Forum

   `http://www.ogf.org/gf/group_info/view.php?group=jsdl-wg`

   The JSDL HPC Profile Application Extension, Version 1.0 is GFD 111:

   `http://www.ogf.org/documents/GFD.111.pdf`

4. OGSA Usage Record Working Group (UR-WG) of the Open Grid Forum

   The Usage Record - Format Recommendation is GFD.98

   `http://www.ogf.org/documents/GFD.98.pdf`

5. Network Working Group, Uniform Resource Identifier (URI) : Generic Syntax

   `http://www.rfc-editor.org/rfc/rfc3986.txt`

Chapter 11

Submitting Cray Jobs

11.1 Introduction

You can submit jobs that are designed to run on the Cray, using the PBS select and place syntax.

11.2 PBS Jobs on the Cray

When you submit a job that is designed to run on the Cray, you create a job script that contains the same `aprun` command as a non-PBS job, but submit the job using the PBS select and place syntax. You can translate the mpp* syntax into select and place syntax using the rules described in section 11.3.2 “Automatic Translation of mpp* Resource Requests” on page 297.

You can submit a PBS job using mpp* syntax, but mpp* syntax is deprecated.

If a job does not request a login node, one is chosen for it. A login node is assigned to each PBS job that runs on the Cray. The job script runs on this login node.

Jobs requesting a vntype of *cray_compute* are expected to have an `aprun` in the job script to launch the job on the compute nodes. PBS does not do any verification that the job script contains an `aprun` statement.

11.3 PBS Resources for the Cray

11.3.1 Built-in and Custom Resources for the Cray

PBS provides built-in and custom resources specifically created for jobs on the Cray. The custom resources are created by PBS to reflect Cray information such as segments or labels.

nchunk

This is the number of chunks requested between plus symbols in a select statement. For example, if the select statement is `-lselect 4:ncpus=2+12:ncpus=8`, the value of nchunk for the first part is *4*, and for the second part it is *12*. The nchunk resource cannot be named in a select statement; it can only be specified as a number preceding the colon, as in the above example. When the number is omitted, nchunk is 1.

Non-consumable.

Settable by Manager and Operator; readable by all.

Format: *Integer*

Python type: int

Default value: *1*

vntype

This resource represents the type of the vnode. Automatically set by PBS to one of two specific values for cray vnodes. Has no meaning for non-Cray vnodes.

Non-consumable.

Format: *String array*

Automatically assigned values for Cray vnodes:

cray_compute

This vnode represents part of a compute node.

cray_login

This vnode represents a login node.

Default value: None

Python type: str

PBScrayhost

On CLE 2.2, this is set to "*default*".

On CLE 3.0 and higher, used to delineate a Cray system, containing ALPS, login nodes running PBS MOMs, and compute nodes, from a separate Cray system with a separate ALPS. Non-consumable. The value of PBScrayhost is set to the value of mpp_host for this system.

Format: *String*

Default: CLE 2.2: "*default*"; CLE 3.0 and higher: None

PBScraylabel_<label name>

Custom resource created by PBS for the Cray. Tracks labels applied to compute nodes. For each label on a compute node, PBS creates a custom resource whose name is a concatenation of *PBScraylabel_* and the name of the label. PBS sets the value of the resource to *True* on all vnodes representing the compute node.

Format: *PBScraylabel_<label name>*

For example, if the label name is *Blue*, the name of this resource is *PBScraylabel_Blue*.

Format: *Boolean*

Default: None

PBScraynid

Custom resource created by PBS for the Cray. Used to track the node ID of the associated compute node. All vnodes representing a particular compute node share a value for PBScraynid. Non-consumable.

The value of PBScraynid is set to the value of node_id for this compute node.

Non-consumable.

Format: *String*

Default: None

PBScrayorder

Custom resource created by PBS for the Cray. Used to track the order in which compute nodes are listed in the Cray inventory. All vnodes associated with a particular compute node share a value for PBScrayorder. Non-consumable.

Vnodes for the first compute node listed are assigned a value of 1 for PBScrayorder. The vnodes for each subsequent compute node listed are assigned a value one greater than the previous value.

Do not use this resource in a resource request.

Format: *Integer*

Default: None

PBScrayseg

Custom resource created by PBS for the Cray. Tracks the segment ordinal of the associated NUMA node. For the first NUMA node of a compute host, the segment ordinal is *0*, and the value of PBScrayseg for the associated vnode is *0*. For the second NUMA node, the segment ordinal is *1*, PBScrayseg is *1*, and so on. Non-consumable.

Format: *String*

Default: None

11.3.2 Automatic Translation of mpp* Resource Requests

When a PBS job or reservation is submitted using the mpp* syntax, PBS translates the mpp* resource request into PBS select and place statements. The translation uses the following rules:

- For each chunk on a vnode representing a compute node, the vntype resource is set to *cray_compute*. (Using mpp* implies the use of compute nodes.)
- If the job requests `-lvnode=<value>`, the following becomes or is added to the equivalent chunk request:

  `:vnode=<value>`
- If the job requests `-lhost=<value>`, the following becomes or is

added to the equivalent chunk request:

`:host=<value>`

- Translating mppwidth:

  When the job requests mppwidth:

  - If mppnppn is specified, the following happen:
    - nchunk (number of chunks) is set to *mppwidth / mppnppn*
    - mpiprocs is set to mppnppn
    - `-lplace=scatter` is added to the request
  - If mppnppn is not specified, the following happen:
    - mppnppn is treated as if it is *1*
    - nchunk (number of chunks) is set to *mppwidth*
    - `-lplace=free` is added to the request

- Translating mppnppn:

  If mppnppn is not specified, it defaults to *1*.

- Translating mppdepth:

  If mppdepth is not specified, it defaults to *1*.

- ncpus is set to *mppdepth * mppnppn*
- If mpphost is specified as a submit argument, PBS adds a custom resource called PBScrayhost to the select statement, requesting the same value as for mpphost.
- The mppnodes resource is translated by PBS into the corresponding vnodes.
- When a job requests mpplabels, PBS adds a custom resource called PBScraylabel_<label name> to each chunk that requests a vnode from the compute node with that label. For example, if the job

requests:

```
-l mppwidth=1,mpplabels=\"small,red\"
```

the translated request is:

```
-l select=1:
  PBScraylabel_small=True:PBScraylabel_red=True
```

- The following table summarizes how each mpp* resource is translated into select and place statements:

Table 11-1: Mapping mpp* Resources to select and place

mpp* Resource	Resulting PBS Resource	How Value of PBS Resource is Derived
mpparch	arch	arch=*mpparch*
mppdepth*mppnppn (mppdepth defaults to *1* if not specified.)	ncpus	ncpus = *mppdepth*mppnppn*
mpphost	PBScrayhost	PBScrayhost=*mpphost*
mpplabels, for example mpplabels =\"red,small\"	PBScraylabel_red =*True* PBScraylabel_small =*True*	PBS creates custom Boolean resources named *PBScraylabel_<label>*, and sets them to *True* on associated vnodes
mppmem	mem	mem=*mppmem*
mppnodes	Corresponding vnodes	PBS uses vnodes representing requested nodes
mppnppn (Defaults to *1* if not specified.)	mpiprocs	mpiprocs=*mppnppn*

Table 11-1: Mapping mpp* Resources to select and place

mpp* Resource		Resulting PBS Resource	How Value of PBS Resource is Derived
mppwidth	mppnppn specified	nchunk	nchunk = *mppwidth/mppnppn* place=*scatter* mpiprocs=mppnppn Example: if mppwidth=*8* and mppnppn=*2*, nchunk=*4*
	mppnppn not specified	nchunk	nchunk = *mppwidth* place=*free* mpiprocs not set Example: if mppwidth=*8*, nchunk=*8*

11.3.2.1 Examples of Mapping mpp* Resources to select and place

Example 1: You want 8 PEs. The `aprun` statement is the following:

```
aprun -n 8
```

The old resource request using mpp* is the following:

```
qsub -l mppwidth=8
```

The translated select and place is the following:

```
qsub -lselect=8:vntype=cray_compute
```

Example 2: You want 8 PEs with only one PE per compute node. The `aprun` statement is the following:

```
aprun -n 8 -N 1
```

The old resource request using mpp* is the following:

```
qsub -lmppwidth=8,mppnppn=1
```

The translated select and place is the following:

```
qsub -lselect=8:ncpus=1:mpiprocs=1:vntype=cray_compute
   -lplace=scatter
```

Example 3: You want 8 PEs with 2 PEs per compute node. This equates to 4 chunks of 2 ncpus per chunk, scattered across different hosts. The `aprun` statement is the following:

```
aprun -n 8 -N 2
```

The old resource request using mpp* is the following:

```
qsub -lmppwidth=8,mppnppn=2
```

The translated select and place is the following:

```
qsub -lselect=4:ncpus=2:mpiprocs=2:vntype=cray_compute
   -lplace=scatter
```

Example 4: Specifying host:

The old resource request using mpp* is the following:

```
qsub -lmppwidth=8,mpphost=examplehost
```

The translated select and place is the following:

```
qsub -lselect=8:PBScrayhost=examplehost
```

Example 5: Specifying labels:

The old resource request using mpp* is the following:

```
-l mppwidth=1,mpplabels=\"small,red\"
```

The translated select and place is the following:

```
-l select=1:
   PBScraylabel_small=True:PBScraylabel_red=True
```

11.3.3 Resource Accounting

Jobs that request only compute nodes are not assigned resources from login nodes. PBS accounting logs do not show any login node resources being used by these jobs.

Jobs that request login nodes are assigned resources from login nodes, and those resources appear in the PBS accounting logs for these jobs.

PBS performs resource accounting on the login nodes, under the control of their MOMs.

Comprehensive System Accounting (CSA) runs on the compute nodes, under the control of the Cray system.

11.4 Rules for Submitting Jobs on the Cray

11.4.1 Always Specify Node Type

If you want your job to run on Cray nodes, you must specify a Cray node type for your job. You do this by requesting a value for the vntype vnode resource. On each vnode on a Cray, the vntype resource includes one of the following values:

cray_login, for a login node

cray_compute, for a compute node

Each chunk of a Cray job that must run on a login node must request a vntype of *cray_login*.

Each chunk of a Cray job that must run on a compute node must request a vntype of *cray_compute*.

Example 1: Request any login node, and two compute-node vnodes. The job is run on the login node selected by the scheduler:

```
qsub -lselect=1:ncpus=2:vntype=cray_login
   +2:ncpus=2:vntype=cray_compute
```

Example 2: Launch a job on a particular login node by specifying the login node vnode name first in the select line. The job script runs on the specified login node:

```
qsub -lselect=1:ncpus=2:vnode=login1
   +2:ncpus=2:vntype=cray_compute
```

For a description of the vntype resource, see "Built-in Resources" on page 350 of the PBS Professional Reference Guide.

11.4.2 Always Reserve Required Vnodes

Always reserve at least as many PEs as you request in your `aprun` statement.

11.4.3 Requesting Login Node Where Job Script Runs

If you request a login node as part of your resource request, the login node resource request must be the first element of the select statement. The job script is run on the login node. If you request more than one login node, the job script runs on the first login node requested.

11.4.4 Login Nodes in PBS Reservations

If the jobs that are to run in a PBS reservation require a particular login node, you must do the following:

- The reservation must request the specific login node
- Each job that will run in the reservation must request the same login node that the reservation requested

11.4.5 Specifying Number of Chunks

You specify the number of chunks by prefixing each chunk request with an integer. If not specified, this integer defaults to *1*. For example, to specify 4 chunks with 2 CPUs each, and 8 chunks with 1 CPU each:

```
qsub -lselect=4:ncpus=2+8:ncpus=1
```

You cannot request the nchunk resource.

11.4.6 Requesting mppnppn Equivalent

If your job requires the equivalent of mppnppn, you can do either of the following:

- When using select and place statements, use the translation information provided in Table 11-1, "Mapping mpp* Resources to select and place," on page 299, and include `-lplace=scatter` in the job request.
- Include mppnppn in the `qsub` line (mppnppn is deprecated.)

11.4.7 Do Not Mix mpp* and select/place

Jobs cannot use both -lmpp* syntax and -lselect/-lplace syntax.

11.5 Techniques for Submitting Cray Jobs

11.5.1 Specifying Number of PEs per NUMA Node

The Cray `aprun -S` option allows you to specify the number of PEs per NUMA node for your job. PBS allows you to make the equivalent request using select and place statements. PBS jobs on the Cray should scatter chunks across vnodes.

To calculate the select and place requirements, do the following:

- Set nchunk equal to *n* (the width) divided by *S* (the number of PEs per NUMA node):

  nchunk = n/S

- Set ncpus equal to *S* (the number PEs per NUMA node):

  ncpus=S

- Set mpiprocs equal to *S* (same as ncpus)

  mpiprocs=S

Example 1: You want a total of 6 PEs with 2 PEs per NUMA node. The `aprun` command is the following:

```
aprun -S 2 -n 6 myjob
```

The equivalent select and place statements are:

```
qsub -lselect=3:ncpus=2:mpiprocs=2 -lplace=vscatter
```

Given two compute nodes, each with two NUMA nodes, where each NUMA node has four PEs, two PEs from each of three of the NUMA nodes are assigned to the job.

Example 2: To request 8 PES, with 4 PEs per NUMA node, the `aprun` statement is the following:

```
aprun -S 4 -n 8
```

The equivalent select statement is the following, not including the scatter-by-vnode and exclusive-by-host placement language:

```
qsub -lselect=2:ncpus=4:mpiprocs=4
```

11.5.1.1 Caveats For `aprun -S`

When you use `aprun -S`, you must request mpiprocs, and request the same value as for ncpus.

11.5.2 Reserving *N* NUMA Nodes Per Compute Node

The Cray `aprun -sn` option allows you to specify the number of NUMA nodes per compute node for your job. PBS allows you to make the equivalent request using select and place statements.

To request *N* NUMA nodes per compute node, you place your job by requesting a resource that specifies the number of of NUMA nodes per compute node. This resource is set up by your administrator. We suggest that the resource is named *craysn*, and the value you specify is the number of vnodes per compute node. For example, to request 2 segments per compute node, specify a value of *2* for craysn.

To make a request equivalent to `aprun -sn 3 -n 24`, and match the compute node exclusive behavior of the Cray, you can specify the following:

```
qsub -lselect=24:ncpus=1:craysn=3 -lplace=exclhost
```

11.5.3 Reserving Specific NUMA Nodes on Each Compute Node

The Cray `aprun -sl` option allows you to reserve specific NUMA nodes on the compute nodes your job uses. PBS allows you to make the equivalent request using select and place statements.

How you request resources depends on the number of NUMA nodes you want per compute node, and how the administrator has set up the resource that allows you to request specific compute nodes.

11.5.3.1 Requesting a Single NUMA Node Per Compute Node

You can request the PBScrayseg resource to request one particular NUMA node per compute node. PBS automatically creates a custom string resource called PBScrayseg, and sets the value for each vnode to be the segment ordinal for the associated NUMA node. See "Custom Cray Resources" on page 357 of the PBS Professional Reference Guide.

Example 1: You want 8 PEs total, using only NUMA node 1 on each compute node. The `aprun` statement is the following:

```
aprun -sl 1 -n 8
```

An equivalent resource request for a PBS job is the following:

```
qsub -lselect=8:ncpus=1:PBScrayseg=1
```

See section 11.3.1 "Built-in and Custom Resources for the Cray" on page 294.

11.5.3.2 Requesting Multiple NUMA Nodes Per Compute Node

If you want to request multiple NUMA nodes per compute node, you have choices. For example, if your `aprun` statement looks like the following:

```
aprun -sl 0,1 -n 8
```

You can do any of the following:

- You can request separate chunks for each segment:

```
qsub -lselect=4:ncpus=1:PBScrayseg=0
   +4:ncpus=1:PBScrayseg=1
```

- If you know about the underlying hardware, the PBS resource request can take advantage of that. On a homogenous system with 2 NUMA nodes per compute node and 4 PEs per NUMA node, you can use the following PBS resource request:

```
qsub -lselect=8:ncpus=1 -lplace=pack
```

- If the administrator has set up a resource that allows you to request NUMA node combinations, called for example *segmentcombo*, you request a value for the resource that is the list of vnodes you want. The equivalent select statement which uses this resource is the following:

```
qsub -lselect=8:ncpus=1:segmentcombo=01 jobscript
```

11.5.3.3 Caveat When Using Combination or Number Resources

You **must** use the same resource string values as the ones set up by the administrator. "*012*" is **not** the same as "*102*" or "*201*".

11.5.4 Requesting Groups of Login Nodes

If you want to use groups of esLogin nodes and internal login nodes, your administrator can set the vntype resource on these nodes to a special value, for example *cray_compile*.

To submit a job requesting any combination of esLogin nodes and internal login nodes, you specify the special value for the vntype resource in your select statement. For example:

```
qsub -lselect=4:ncpus=1:vntype=cray_compile job
```

11.5.5 Using Internal Login Nodes Only

Compiling, preprocessing, and postprocessing jobs can run on internal login nodes. Internal login nodes have a vntype value of *cray_login*. If you want to run a job that needs to use the resources on internal login nodes only, you can specify `vntype=cray_login` in your select statement. For example:

```
qsub -lselect=4:ncpus=1:vntype=cray_login job
```

11.5.6 Using Compute Nodes

If your job script contains an `aprun` launch, you must run your job on compute nodes. To run your job on compute nodes, specify a vntype of *cray_compute*. For example:

```
lselect=2:ncpus=2:vntype=cray_compute
```

11.5.7 Using Login and Compute Nodes

You can request both login and compute nodes for your job. You must specify the login node(s) before the compute nodes. You can specify a vntype of *cray_login* for the chunks requiring login nodes, and a vntype of *cray_compute* for the chunks requiring compute nodes. For example:

```
qsub -lselect=1:ncpus=2:vntype=cray_login
   +2:ncpus=2:vntype=cray_compute
```

11.5.8 Requesting Specific Groups of Nodes

You can use select and place to request the groups of vnodes you want. This replaces the behavior provided by mppnodes.

Users may need to group their nodes by the some criteria, for example:

- Certain nodes are fast nodes
- Certain nodes share a required or useful characteristic
- Some combination of nodes gives the best performance for an application

Your administrator can set up either of the following:

- Custom Boolean resources on each vnode, which reflect how the nodes are labeled, and allow you to request the vnodes that represent the group of nodes you want. These resources are named *PBScraylabel_<label name>,* and set to *True* on the vnodes that represent the labeled nodes.

  Your administrator must label the groups of nodes. For example, if a node is both fast and best for App1, it can have two labels, *fast*, and *BestForApp1*.

  To request the fast nodes in this example, add the following to each chunk request:

  :PBScraylabel_fast=True

- Other custom resources on each vnode, which are set to reflect the vnode's characteristics. For example, if a vnode is fast, it can have a custom string resource called "*speed*", with a value of *fast* on that vnode. You must ask your administrator for the name and possible values for the resource.

11.5.9 Requesting Nodes in Specific Order

Your application may perform better when the ranks are laid out on specific nodes in a specific order. If you want to request vnodes so that the nodes are in a specific order, you can specify the host for each chunk of the job. For example, if you need nodes ordered "nid0, nid2, nid4", you can request the following:

```
qsub -l select=2:ncpus=2:host=nid0 +2:ncpus=2:host=nid2
   +2:ncpus=2:host=nid4
```

11.5.10 Requesting Interlagos Hardware

PBS allows you to specifically request (or avoid) Interlagos hardware. Your administrator must create a Boolean resource on each vnode, and set it to *True* where the vnode has Interlagos hardware. We recommend that the Boolean is called "*PBScraylabel_interlagos*".

You request or avoid this resource using `PBScraylabel_interlagos=True` or `PBScraylabel_interlagos=False`. For example:

```
qsub -lselect=3:ncpus=2:PBScraylabel_interlagos=true
   myjob
```

11.6 Viewing Cray Job Information

11.6.1 Finding Out Where Job Was Launched

To determine the internal login node where the job was launched, use the `qstat -f` command:

```
qstat -f <job ID>
```

Look at the exec_host line of the output. The first vnode is the login node where the job was launched.

11.6.2 Finding Out How mpp* Request Was Translated

- To find out how the mpp* job request was translated into select and place statements, use the `qstat -f` command:

  ```
  qstat -f[x] <job ID>
  ```

  Look at the Resource_List.select job attribute. The original is in the Submit_arguments job attribute.

- To find out how the mpp* reservation request was translated into select

and place statements, use the pbs_rstat command:

```
pbs_rstat -F <reservation ID>
```

Look at the Resource_List attribute.

11.6.3 Viewing Original mpp* Request

To see the original mpp* request, use the qstat command:

```
qstat -f[x] <job ID>
```

The Submit_arguments field contains the original mpp* request.

11.6.4 Listing Jobs Running on Vnode

To see which jobs are running on a vnode, use the pbsnodes command:

```
pbsnodes -av
```

The jobs attribute of each vnode lists the jobs running on that vnode. Jobs launched from an internal login node, requesting a vntype of *cray_compute* only, are not listed in the internal login node's vnode's jobs attribute. Jobs that are actually running on a login node, which requested a vntype of *cray_login*, do appear in the login node's vnode's jobs attribute.

11.6.4.1 Caveats When Listing Jobs

Jobs that requested a vntype of *cray_compute* that were launched from an internal login node are not listed in the jobs attribute of the internal login node.

11.6.4.2 Example Output

Example of pbsnodes -av output:

```
examplehost_36_0
   Mom = host1.example.com,host2.example.com
   ntype = PBS
   state = free
   resources_available.arch = XT
   resources_available.PBScrayhost = examplehost
   resources_available.PBScraynid = 36
   resources_available.PBScrayseg = 0
   resources_available.PBScrayorder = 1
   resources_available.host = examplehost_36
   resources_available.mem = 2048000kb
   resources_available.ncpus = 2
   resources_available.vnode = examplehost_36_0
   resources_available.vntype = cray_compute
   resources_assigned.mem = 0kb
   resources_assigned.ncpus = 0
   resources_assigned.netwins = 0
   resources_assigned.vmem = 0kb
   resv_enable = True
   sharing = default_shared
```

11.6.5 How ALPS Request Is Constructed

The reservation request that is sent to the Cray is constructed from the contents of the exec_vnode and Resource_List.select job attributes. If the exec_vnode attribute contains chunks asking for the same ncpus and

mem, these are grouped into one section of an ALPS request. Cray requires one CPU per thread. The ALPS request is constructed using the following rules:

Table 11-2: How Cray Elements Are Derived From exec_vnode Terms

Cray Element	exec_vnode Term
Processing Element (PE)	mpiprocs
Requested number of PEs / compute node in this section of job request (width)	Total mpiprocs on vnodes representing compute node involved in this section of job request
Number of threads per PE (depth)	(total assigned ncpus on vnodes representing a compute node) / (total mpiprocs on vnodes representing a compute node)
Memory per PE (mem)	(total memory in chunk request)/total mpiprocs in chunk
Number of PEs per compute node (nppn)	Sum of mpiprocs on vnodes representing a compute node
Number of PEs per segment (npps)	Not used.
Number of segments per node (nspn)	Not used.
NUMA node (segments)	Not used.

11.7 Caveats and Advice

11.7.1 Use select and place Instead of mpp*

It is recommended to use select and place instead of mpp* resources. The mpp* resources are deprecated.

11.7.2 Using Combination or Number Resources

When requesting a resource that is set up by the administrator, you **must** use the same resource string values as the ones set up by the administrator. "*012*" is **not** the same as "*102*" or "*201*". For example, when requesting a resource that allows you to request NUMA nodes 0 and 1, and the administrator used the string *01*, you must request `<resource name>=01`. If you request `<resource name>=10`, this will not work.

11.7.3 Avoid Invalid Cray Requests

It is possible to create a select and place statement that meets the requirements of PBS but not of the Cray.

Example 1: The Cray width and depth values cannot be calculated from ncpus and mpiprocs values. For example, if ncpus is 2 and mpiprocs is 4, the depth value is calculated by dividing ncpus by mpiprocs, and is one-half. This is not a valid depth value for Cray.

Example 2: ALPS cannot run jobs with some complex select statements. In particular, a multiple program, multiple data (MPMD) ALPS reservation where two groups span a compute node will produce an ALPS error, because the nid shows up in two ReserveParam sections.

11.7.4 Visibility of Jobs Launched from Login Nodes

Jobs that requested a vntype of *cray_compute* that were launched from an internal login node are not listed in the jobs attribute of the internal login node.

11.7.5 Resource Restrictions and Deprecations

11.7.5.1 Restriction on Translation of mpp* Resources

PBS translates only the following mpp* resources into select and place syntax:

mppwidth

mppdepth

mppnppn

mppmem

mpparch

mpphost

mpplabels

mppnodes

11.7.5.2 mpp* Resources Deprecated

The mpp* syntax is deprecated. See section 1.3, "Deprecations" on page 10 in the PBS Professional Administrator's Guide.

11.7.6 Do Not Mix mpp* and select/place

Jobs cannot use both -lmpp* syntax and -lselect/-lplace syntax.

11.7.7 Do Not Request PBScrayorder

Do not use PBScrayorder in a resource request.

11.8 Errors and Logging

11.8.1 Invalid Cray Requests

When a select statement does not meet Cray requirements, and the Cray reservation fails, the following error message is printed in MOM's log, at log event class 0x080:

"Fatal MPP reservation error preparing request"

11.8.2 Job Requests More Than Available

If do_not_span_psets is set to *True*, and a job requests more resources than are available in one placement set, the following happens:

- The job's comment is set to the following:

  "Not Running: can't fit in the largest placement set, and can't span psets"

- The following message is printed to the scheduler's log:

  "Can't fit in the largest placement set, and can't span placement sets"

11.8.3 All Requested mppnodes Not Found

If mppnodes are requested, but there are no vnodes that match the requested mppnodes (i.e. 0% of the mppnodes list is found), the job or reservation is rejected with the following message:

"The following error was encountered: No matching vnodes for the given mppnodes <mppnodes>"

A log message is printed to the server log at event class 0x0004:

"translate mpp: ERROR: could not find matching vnodes for the given mppnodes <mppnodes (as input)>"

11.8.4 Some Requested mppnodes Not Found

If mppnodes are requested, and only some of the mppnodes are found to match the vnodes, then the job or reservation is accepted, but the following is printed in the server log at event class 0x0004:

"translate mpp: could not find matching vnodes for these given mppnodes [<comma separated list of mppnodes>]"

The job may or may not run depending on whether the vnodes that were matched up to the requested mppnodes have enough resources for the job.

11.8.5 Bad mppnodes Range

If the resource request specifies an mppnodes range with the value on the right hand side of the range less than or equal to the value on the left hand side of the range, the job or reservation is rejected with the following message:

```
The following error was encountered:
Bad range '<range>', the first number (<left_side>)
   must be less than the second number (<right_side>)
```

A log message is printed to the server log at event class 0x0004:

"translate mpp: ERROR: bad range '<range>', the first number (<left_side>) must be less than the second number (<right_side>)"

11.8.6 Resource Request Containing Both mpp* and select/place

If a resource request contains both mpp* and select/place, the job or reservation is rejected, and the following error is printed:

"The following error was encountered:
mpp resources cannot be used with "select" or "place""

Chapter 12

Using Provisioning

PBS provides automatic provisioning of an OS or application on vnodes that are configured to be provisioned. When a job requires an OS that is available but not running, or an application that is not installed, PBS provisions the vnode with that OS or application.

12.1 Definitions

AOE

The environment on a vnode. This may be one that results from provisioning that vnode, or one that is already in place

Provision

To install an OS or application, or to run a script which performs installation and/or setup

Provisioned Vnode

A vnode which, through the process of provisioning, has an OS or application that was installed, or which has had a script run on it

12.2 How Provisioning Works

Provisioning can be performed only on vnodes that have provisioning enabled, shown in the vnode's provision_enable attribute.

Provisioning can be the following:

- Directly installing an OS or application
- Running a script which may perform setup or installation

Each vnode is individually configured for provisioning with a list of available AOEs, in the vnode's resources_available.aoe attribute.

Each vnode's current_aoe attribute shows that vnode's current AOE. The scheduler queries each vnode's aoe resource and current_aoe attribute in order to determine which vnodes to provision for each job.

Provisioning can be used for interactive jobs.

A job's walltime clock starts when provisioning for the job has finished.

12.2.1 Causing Vnodes To Be Provisioned

An AOE can be requested for a job or a reservation. When a job requests an AOE, that means that the job will be run on vnodes running that AOE. When a reservation requests an AOE, that means that the reservation reserves vnodes that have that AOE available. The AOE is instantiated on reserved vnodes only when a job requesting that AOE runs.

When the scheduler runs each job that requests an AOE, it either finds the vnodes that satisfy the job's requirements, or provisions the required vnodes. For example, if SLES is available on a set of vnodes that other-

wise suit your job, you can request SLES for your job, and regardless of the OS running on those vnodes before your job starts, SLES will be running at the time the job begins execution.

12.2.2 Using an AOE

When you request an AOE for a job, the requested AOE must be one of the AOEs that has been configured at your site. For example, if the AOEs available on vnodes are "*rhel*" and "*sles*", you can request only those; you cannot request "*suse*".

You can request a reservation for vnodes that have a specific AOE available. This way, jobs needing that AOE can be submitted to that reservation. This means that jobs needing that AOE are guaranteed to be running on vnodes that have that AOE available.

Each reservation can have at most one AOE specified for it. Any jobs that run in that reservation must not request a different AOE from the one requested for the reservation. That is, the job can run in the reservation if it either requests no AOE, or requests the same AOE as the reservation.

12.2.3 Job Substates and Provisioning

When a job is in the process of provisioning, its substate is *provisioning*. This is the description of the substate:

provisioning
: The job is waiting for vnode(s) to be provisioned with its requested AOE. Integer value is *71*. See "Job Substates" on page 464 of the PBS Professional Reference Guide for a list of job substates.

The following table shows how provisioning events affect job states and substates:

Table 12-1: Provisioning Events and Job States/Substates

Event	Initial Job State, Substate	Resulting Job State, Substate
Job submitted		*Queued and ready for selection*
Provisioning starts	*Queued, Queued*	*Running, Provisioning*
Provisioning fails to start	*Queued, Queued*	*Held, Held*
Provisioning fails	*Running, Provisioning*	*Queued, Queued*
Provisioning succeeds and job runs	*Running, Provisioning*	*Running, Running*
Internal error occurs	*Running, Provisioning*	*Held, Held*

12.3 Requirements and Restrictions

12.3.1 Host Restrictions

12.3.1.1 Single-vnode Hosts Only

PBS will provision only single-vnode hosts. Do not attempt to use provisioning on hosts that have more than one vnode.

12.3.1.2 Server Host Cannot Be Provisioned

The Server host cannot be provisioned: a MOM can run on the Server host, but that MOM's vnode cannot be provisioned. The provision_enable vnode attribute, resources_available.aoe, and current_aoe cannot be set on the Server host.

12.3.2 AOE Restrictions

Only one AOE can be instantiated at a time on a vnode.

Only one kind of aoe resource can be requested in a job. For example, an acceptable job could make the following request:

```
-l select=1:ncpus=1:aoe=suse+1:ncpus=2:aoe=suse
```

12.3.2.1 Vnode Job Restrictions

A vnode with any of the following jobs will not be selected for provisioning:

- One or more running jobs
- A suspended job
- A job being backfilled around

12.3.2.2 Provisioning Job Restrictions

A job that requests an AOE will not be backfilled around.

12.3.2.3 Vnode Reservation Restrictions

A vnode will not be selected for provisioning for job MyJob if the vnode has a confirmed reservation, and the start time of the reservation is before job MyJob will end.

A vnode will not be selected for provisioning for a job in reservation R1 if the vnode has a confirmed reservation R2, and an occurrence of R1 and an occurrence of R2 overlap in time and share a vnode for which different AOEs are requested by the two occurrences.

12.3.3 Requirements for Jobs

12.3.3.1 If AOE is Requested, All Chunks Must Request Same AOE

If any chunk of a job requests an AOE, all chunks must request that AOE.

If a job requesting an AOE is submitted to a reservation, that reservation must also request the same AOE.

12.4 Using Provisioning

12.4.1 Requesting Provisioning

You request a reservation with an AOE in order to reserve the resources and AOE required to run a job. You request an AOE for a job if that job requires that AOE. You request provisioning for a job or reservation using the same syntax.

You can request an AOE for the entire job/reservation:

-l aoe = <AOE>

Example:

```
-l aoe = suse
```

The *-l <AOE>* form cannot be used with *-l select.*

You can request an AOE for a single-chunk job/reservation:

-l select=<chunk request>:aoe=<AOE>

Example:

```
-ls select=1:ncpus=2:aoe=rhel
```

You can request the same AOE for each chunk of a job/reservation:

-l select=<chunk request>:aoe=<AOE> + <chunk request>:aoe=<AOE>

Example:

```
-l select=1:ncpus=1:aoe=suse + 2:ncpus=2:aoe=suse
```

12.4.2 Commands and Provisioning

If you try to use PBS commands on a job that is in the *provisioning* substate, the commands behave differently. The provisioning of vnodes is not affected by the commands; if provisioning has already started, it will continue. The following table lists the affected commands:

Table 12-2: Effect of Commands on Jobs in Provisioning Substate

Command	Behavior While in Provisioning Substate
`qdel`	(Without force) Job is not deleted
	(With force) Job is deleted
`qsig -s suspend`	Job is not suspended
`qhold`	Job is not held
`qrerun`	Job is not requeued

Table 12-2: Effect of Commands on Jobs in Provisioning Substate

Command	Behavior While in Provisioning Substate
`qmove`	Cannot be used on a job that is provisioning
`qalter`	Cannot be used on a job that is provisioning
`qrun`	Cannot be used on a job that is provisioning

12.4.3 How Provisioning Affects Jobs

A job that has requested an AOE will not preempt another job. Therefore no job will be terminated in order to run a job with a requested AOE.

A job that has requested an AOE will not be backfilled around.

12.5 Caveats and Errors

12.5.1 Requested Job AOE and Reservation AOE Should Match

Do not submit jobs that request an AOE to a reservation that does not request the same AOE. Reserved vnodes may not supply that AOE; your job will not run.

12.5.2 Allow Enough Time in Reservations

If a job is submitted to a reservation with a duration close to the walltime of the job, provisioning could cause the job to be terminated before it finishes running, or to be prevented from starting. If a reservation is designed to take jobs requesting an AOE, leave enough extra time in the reservation for provisioning.

12.5.3 Requesting Multiple AOEs For a Job or Reservation

Do not request more than one AOE per job or reservation. The job will not run, or the reservation will remain unconfirmed.

12.5.4 Held and Requeued Jobs

The job is held with a system hold for the following reasons:

- Provisioning fails due to invalid provisioning request or to internal system error
- After provisioning, the AOE reported by the vnode does not match the AOE requested by the job

The hold can be released by the PBS Administrator after investigating what went wrong and correcting the mistake.

The job is requeued for the following reasons:

- The provisioning hook fails due to timeout
- The vnode is not reported back up

12.5.5 Conflicting Resource Requests

The values of the resources arch and vnode may be changed by provisioning. Do not request an AOE and either arch or vnode for the same job.

12.5.6 Job Submission and Alteration Have Same Requirements

Whether you use the `qsub` command to submit a job, or the `qalter` command to alter a job, the job must eventually meet the same requirements. You cannot submit a job that meets the requirements, then alter it so that it does not.

Appendix A: Converting NQS to PBS

For those converting to PBS from NQS or NQE, PBS includes a utility called **nqs2pbs** which converts an existing NQS job script so that it will work with PBS. (In fact, the resulting script will be valid to both NQS and PBS.) The existing script is copied and PBS directives ("#PBS") are inserted prior to each NQS directive (either "#QSUB" or "#Q$") in the original script.

```
nqs2pbs existing-NQS-script new-PBS-script
```

Section "Setting Up Your UNIX/Linux Environment" on page 14 discusses PBS environment variables.

A queue complex in NQS was a grouping of queues within a batch Server. The purpose of a complex was to provide additional control over resource usage. The advanced scheduling features of PBS eliminate the requirement for queue complexes.

13.1 Converting Date Specifications

Converting NQS date specifications to the PBS form may result in a warning message and an incomplete converted date. PBS does not support date specifications of “today”, “tomorrow”, or the name of the days of the week such as “Monday”. If any of these are encountered in a script, the PBS specification will contain only the time portion of the NQS specification (i.e. `#PBS -a hhmm[.ss]`). It is suggested that you specify the execution time on the `qsub` command line rather than in the script. All times are taken as local time. If any unrecognizable NQS directives are encountered, an error message is displayed. The new PBS script will be deleted if any errors occur.

Appendix B: License Agreement

CAUTION!

PRIOR TO INSTALLATION OR USE OF THE SOFTWARE YOU MUST CONSENT TO THE FOLLOWING SOFTWARE LICENSE TERMS AND CONDITIONS BY CLICKING THE “I ACCEPT” BUTTON BELOW. YOUR ACCEPTANCE CREATES A BINDING LEGAL AGREEMENT BETWEEN YOU AND ALTAIR. IF YOU DO NOT HAVE THE AUTHORITY TO BIND YOUR ORGANIZATION TO THESE TERMS AND CONDITIONS, YOU MUST CLICK “I DO NOT ACCEPT” AND THEN HAVE AN AUTHORIZED PARTY IN THE ORGANIZATION THAT YOU REPRESENT ACCEPT THESE TERMS.

IF YOU, OR THE ORGANIZATION THAT YOU REPRESENT, HAS A MASTER SOFTWARE LICENSE AGREEMENT (“MASTER SLA”) ON FILE AT THE CORPORATE HEADQUARTERS OF ALTAIR ENGI-

NEERING, INC. (“ALTAIR”), THE MASTER SLA TAKES PRECEDENCE OVER THESE TERMS AND SHALL GOVERN YOUR USE OF THE SOFTWARE.

MODIFICATION(S) OF THESE SOFTWARE LICENSE TERMS IS EXPRESSLY PROHIBITED. ANY ATTEMTED MODIFICATION(S) WILL BE NONBINDING AND OF NO FORCE OR EFFECT UNLESS EXPRESSLY AGREED TO IN WRITING BY AN AUTHORIZED CORPORATE OFFICER OF ALTAIR. ANY DISPUTE RELATING TO THE VALIDITY OF AN ALLEGED MODIFICATION SHALL BE DETERMINED IN ALTAIR’S SOLE DISCRETION.

Altair Engineering, Inc. - Software License Agreement

THIS SOFTWARE LICENSE AGREEMENT, including any Additional Terms (together with the “Agreement”), shall be effective as of the date of YOUR acceptance of these software license terms and conditions (the “Effective Date”) and is between ALTAIR ENGINEERING, INC., 1820 E. Big Beaver Road, Troy, MI 48083-2031, USA, a Michigan corporation (“Altair”), and YOU, or the organization on whose behalf you have authority to accept these terms (the “Licensee”). Altair and Licensee, intending to be legally bound, hereby agree as follows:

1. DEFINITIONS. In addition to terms defined elsewhere in this Agreement, the following terms shall have the meanings defined below for purposes of this Agreement:

Additional Terms. Additional Terms are those terms and conditions which are determined by an Altair Subsidiary to meet local market conditions.

Documentation. Documentation provided by Altair or its resellers on any media for use with the Products.

Execute. To load Software into a computer's RAM or other primary memory for execution by the computer.

Global Zone: Software is licensed based on three Global Zones: the Americas, Europe and Asia-Pacific. When Licensee has Licensed Workstations located in multiple Global Zones, which are connected to a single License (Network) Server, a premium is applied to the standard Software License pricing for a single Global Zone.

ISV/Independent Software Vendor. A software company providing its products, (“ISV Software”) to Altair's Licensees through the Altair License Management System using Altair License Units.

License Log File. A computer file providing usage information on the Software as gathered by the Software.

License Management System. The license management system (LMS) that accompanies the Software and limits its use in accordance with this Agreement, and which includes a License Log File.

License (Network) Server. A network file server that Licensee owns or leases located on Licensee's premises and identified by machine serial number and/or HostID on the Order Form.

License Units. A parameter used by the LMS to determine usage of the Software permitted under this Agreement at any one time.

Licensed Workstations. Single-user computers located in the same Global Zone(s) that Licensee owns or leases that are connected to the License (Network) Server via local area network or Licensee's private wide-area network.

Maintenance Release. Any release of the Products made generally available by Altair to its Licensees with annual leases, or those with perpetual licenses who have an active maintenance agreement in effect, that corrects programming errors or makes other minor changes to the Software. The fees for maintenance and support services are included in the annual license fee but perpetual licenses require a separate fee.

Order Form. Altair's standard form in either hard copy or electronic format that contains the specific parameters (such as identifying Licensee's contracting office, License Fees, Software, Support, and License (Network) Servers) of the transaction governed by this Agreement.

Products. Products include Altair Software, ISV Software, and/or Suppliers' software; and Documentation related to all of the forgoing.

Proprietary Rights Notices. Patent, copyright, trademark or other proprietary rights notices applied to the Products, packaging or media.

Software. The Altair software identified in the Order Form and any Updates or Maintenance Releases.

Subsidiary. Subsidiary means any partnership, joint venture, corporation or other form of enterprise in which a party possesses, directly or indirectly, an ownership interest of fifty percent (50%) or greater, or managerial or operational control.

Suppliers. Any person, corporation or other legal entity which may provide software or documents which are included in the Software.

Support. The maintenance and support services provided by Altair pursuant to this Agreement.

Templates. Human readable ASCII files containing machine-interpretable commands for use with the Software.

Term. The term of licenses granted under this Agreement. Annual licenses shall have a 12-month term of use unless stated otherwise on the Order Form. Perpetual licenses shall have a term of twenty-five years. Maintenance agreements for perpetual licenses have a 12-month term.

Update. A new version of the Products made generally available by Altair to its Licensees that includes additional features or functionalities but is substantially the same computer code as the existing Products.

2. LICENSE GRANT. Subject to the terms and conditions set forth in this Agreement, Altair hereby grants Licensee, and Licensee hereby accepts, a limited, non-exclusive, non-transferable license to: a) install the Products on the License (Network) Server(s) identified on the Order Form for use only at the sites identified on the Order Form; b) execute the Products on Licensed Workstations in accordance with the LMS for use solely by Licensee's employees, or its onsite Contractors who have agreed to be bound by the terms of this Agreement, for Licensee's internal business use on Licensed Workstations within the Global Zone(s) as identified on the Order Form and for the term identified on the Order Form; c) make backup copies of the Products, provided that Altair's and its Suppliers' and ISV's Proprietary Rights Notices are reproduced on each such backup copy; d) freely modify and use Templates, and create interfaces to Licensee's proprietary software for internal use only using APIs provided that such modifications shall not be subject to Altair's warranties, indemnities, support or other Altair obligations under this Agreement; and e) copy and distribute Documentation inside Licensee's organization exclusively for use by Licensee's employees and its onsite Contractors who have agreed to be bound by the terms of this Agreement. A copy of the License Log File shall be made available to Altair automatically on no less than a monthly basis. In

the event that Licensee uses a third party vendor for information technology (IT) support, the IT company shall be permitted to access the Software only upon its agreement to abide by the terms of this Agreement. Licensee shall indemnify, defend and hold harmless Altair for the actions of its IT vendor(s).

3. RESTRICTIONS ON USE. Notwithstanding the foregoing license grant, Licensee shall not do (or allow others to do) any of the following: a) install, use, copy, modify, merge, or transfer copies of the Products, except as expressly authorized in this Agreement; b) use any back-up copies of the Products for any purpose other than to replace the original copy provided by Altair in the event it is destroyed or damaged; c) disassemble, decompile or "unlock", reverse translate, reverse engineer, or in any manner decode the Software or ISV Software for any reason; d) sublicense, sell, lend, assign, rent, distribute, publicly display or publicly perform the Products or Licensee's rights under this Agreement; e) allow use outside the Global Zone(s) or User Sites identified on the Order Form; f) allow third parties to access or use the Products such as through a service bureau, wide area network, Internet location or time-sharing arrangement except as expressly provided in Section 2(b); g) remove any Proprietary Rights Notices from the Products; h) disable or circumvent the LMS provided with the Products; or (i) link any software developed, tested or supported by Licensee or third parties to the Products (except for Licensee's own proprietary software solely for Licensee's internal use).

4. OWNERSHIP AND CONFIDENTIALITY. Licensee acknowledges that all applicable rights in patents, copyrights, trademarks, service marks, and trade secrets embodied in the Products are owned by Altair and/or its Suppliers or ISVs. Licensee further acknowledges that the Products, and all copies thereof, are and shall remain the sole and exclusive property of Altair and/or its Suppliers and ISVs. This Agreement is a license and not a sale of the Products. Altair retains all rights in the Products not expressly granted to Licensee herein. Licensee acknowledges that the Products are confidential and constitute valuable assets and trade secrets of Altair and/or its Suppliers and ISVs. Licensee agrees to take the same precautions necessary to protect and maintain the confidentiality of the Products as it does to protect its own information of a confidential nature but in any event, no less than a reasonable degree of care, and shall not disclose or make them available to any person or entity except as expressly provided in this Agreement. Licensee shall promptly notify Altair in the event any unauthorized person obtains access to the Products. If Licensee is required by any governmental authority or court of law to disclose Altair's or its ISV's

or its Suppliers' confidential information, then Licensee shall immediately notify Altair before making such disclosure so that Altair may seek a protective order or other appropriate relief. Licensee's obligations set forth in Section 3 and Section 4 of this Agreement shall survive termination of this Agreement for any reason. Altair's Suppliers and ISVs, as third party beneficiaries, shall be entitled to enforce the terms of this Agreement directly against Licensee as necessary to protect Supplier's intellectual property or other rights.

Altair and its resellers providing support and training to licensed end users of the Products shall keep confidential all Licensee information provided to Altair in order that Altair may provide Support and training to Licensee. Licensee information shall be used only for the purpose of assisting Licensee in its use of the licensed Products. Altair agrees to take the same precautions necessary to protect and maintain the confidentiality of the Licensee information as it does to protect its own information of a confidential nature but in any event, no less than a reasonable degree of care, and shall not disclose or make them available to any person or entity except as expressly provided in this Agreement.

5. MAINTENANCE AND SUPPORT. **Maintenance**. Altair will provide Licensee, at no additional charge for annual licenses and for a maintenance fee for paid-up licenses, with Maintenance Releases and Updates of the Products that are generally released by Altair during the term of the licenses granted under this Agreement, except that this shall not apply to any Term or Renewal Term for which full payment has not been received. Altair does not promise that there will be a certain number of Updates (or any Updates) during a particular year. If there is any question or dispute as to whether a particular release is a Maintenance Release, an Update or a new product, the categorization of the release as determined by Altair shall be final. Licensee agrees to install Maintenance Releases and Updates promptly after receipt from Altair. Maintenance Releases and Updates are subject to this Agreement. Altair shall only be obligated to provide support and maintenance for the most current release of the Software and the most recent prior release. **Support**. Altair will provide support via telephone and email to Licensee at the fees, if any, as listed on the Order Form. If Support has not been procured for any period of time for paid-up licenses, a reinstatement fee shall apply. Support consists of responses to questions from Licensee's personnel related to the use of the then-current and most recent prior release version of the Software. Licensee agrees to provide Altair with sufficient information to resolve technical issues as may be reasonably requested by Altair. Licensee agrees to the best of its

abilities to read, comprehend and follow operating instructions and procedures as specified in, but not limited to, Altair's Documentation and other correspondence related to the Software, and to follow procedures and recommendations provided by Altair in an effort to correct problems. Licensee also agrees to notify Altair of a programming error, malfunction and other problems in accordance with Altair's then current problem reporting procedure. If Altair believes that a problem reported by Licensee may not be due to an error in the Software, Altair will so notify Licensee. Questions must be directed to Altair's specially designated telephone support numbers and email addresses. Support will also be available via email at Internet addresses designated by Altair. Support is available Monday through Friday (excluding holidays) from 8:00 a.m. to 5:00 p.m local time in the Global Zone where licensed, unless stated otherwise on the Order Form. **Exclusions**. Altair shall have no obligation to maintain or support (a) altered, damaged or Licensee-modified Software, or any portion of the Software incorporated with or into other software not provided by Altair; (b) any version of the Software other than the current version of the Software or the immediately prior release of the Software; (c) problems caused by Licensee's negligence, abuse or misapplication of Software other than as specified in the Documentation, or other causes beyond the reasonable control of Altair; or (d) Software installed on any hardware, operating system version or network environment that is not supported by Altair. Support also **excludes** configuration of hardware, non- Altair Software, and networking services; consulting services; general solution provider related services; and general computer system maintenance.

6. WARRANTY AND DISCLAIMER. Altair warrants for a period of ninety (90) days after Licensee initially receives the Software that the Software will perform under normal use substantially as described in then current Documentation. Supplier software included in the Software and ISV Software provided to Licensee shall be warranted as stated by the Supplier or the ISV. Copies of the Suppliers' and ISV's terms and conditions of warranty are available on the Altair Support website. Support services shall be provided in a workmanlike and professional manner, in accordance with the prevailing standard of care for consulting support engineers at the time and place the services are performed.

ALTAIR DOES NOT REPRESENT OR WARRANT THAT THE PRODUCTS WILL MEET LICENSEE'S REQUIREMENTS OR THAT THEIR OPERATION WILL BE UNINTERRUPTED OR ERROR-FREE, OR THAT IT WILL BE COMPATIBLE WITH ANY PARTICULAR HARDWARE OR SOFTWARE. ALTAIR

EXCLUDES AND DISCLAIMS ALL EXPRESS AND IMPLIED WARRANTIES NOT STATED HEREIN, INCLUDING THE IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NON-INFRINGEMENT. THE ENTIRE RISK FOR THE PERFORMANCE, NON-PERFORMANCE OR RESULTS OBTAINED FROM USE OF THE PRODUCTS RESTS WITH LICENSEE AND NOT ALTAIR. ALTAIR MAKES NO WARRANTIES WITH RESPECT TO THE ACCURACY, COMPLETENESS, FUNCTIONALITY, SAFETY, PERFORMANCE, OR ANY OTHER ASPECT OF ANY DESIGN, PROTOTYPE OR FINAL PRODUCT DEVELOPED BY LICENSEE USING THE PRODUCTS.

7. INDEMNITY. Altair will defend and indemnify, at its expense, any claim made against Licensee based on an allegation that the Software infringes a patent or copyright ("Claim"); provided, however, that this indemnification does not include claims which are based on Supplier software or ISV software, and that Licensee has not materially breached the terms of this Agreement, Licensee notifies Altair in writing within ten (10) days after Licensee first learns of the Claim; and Licensee cooperates fully in the defense of the claim. Altair shall have sole control over such defense; provided, however, that it may not enter into any settlement binding upon Licensee without Licensee's consent, which shall not be unreasonably withheld. If a Claim is made, Altair may modify the Software to avoid the alleged infringement, provided however, that such modifications do not materially diminish the Software's functionality. If such modifications are not commercially reasonable or technically possible, Altair may terminate this Agreement and refund to Licensee the prorated license fee that Licensee paid for the then current Term. Perpetual licenses shall be pro-rated over a 36-month term. Altair shall have no obligation under this Section 7, however, if the alleged infringement arises from Altair's compliance with specifications or instructions prescribed by Licensee, modification of the Software by Licensee, use of the Software in combination with other software not provided by Altair and which use is not specifically described in the Documentation, and if Licensee is not using the most current version of the Software, if such alleged infringement would not have occurred except for such exclusions listed here. This section 7 states Altair's entire liability to Licensee in the event a Claim is made. No indemnification is made for Supplier and/or ISV Software.

8. LIMITATION OF REMEDIES AND LIABILITY. Licensee's exclusive remedy (and Altair's sole liability) for Software that does not meet the warranty set forth in Section 6 shall be, at Altair's option, either (i) to correct the nonconforming Software within a reasonable time so that it conforms to the warranty; or (ii) to terminate this Agreement and refund to Licensee the license fees that Licensee has paid for the then current Term for the nonconforming Software; provided, however that Licensee notifies Altair of the problem in writing within the applicable Warranty Period when the problem first occurs. Any corrected Software shall be warranted in accordance with Section 6 for ninety (90) days after delivery to Licensee. The warranties hereunder are void if the Software has been improperly installed, misused, or if Licensee has violated the terms of this Agreement.

Altair's entire liability for all claims arising under or related in any way to this Agreement (regardless of legal theory), shall be limited to direct damages, and shall not exceed, in the aggregate for all claims, the license and maintenance fees paid under this Agreement by Licensee in the 12 months prior to the claim on a prorated basis, except for claims under Section 7. ALTAIR AND ITS SUPPLIERS AND ISVS SHALL NOT BE LIABLE TO LICENSEE OR ANYONE ELSE FOR INDIRECT, SPECIAL, INCIDENTAL, OR CONSEQUENTIAL DAMAGES ARISING HEREUNDER (INCLUDING LOSS OF PROFITS OR DATA, DEFECTS IN DESIGN OR PRODUCTS CREATED USING THE SOFTWARE, OR ANY INJURY OR DAMAGE RESULTING FROM SUCH DEFECTS, SUFFERED BY LICENSEE OR ANY THIRD PARTY) EVEN IF ALTAIR OR ITS SUPPLIERS OR ITS ISVS HAVE BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES. Licensee acknowledges that it is solely responsible for the adequacy and accuracy of the input of data, including the output generated from such data, and agrees to defend, indemnify, and hold harmless Altair and its Suppliers and ISVs from any and all claims, including reasonable attorney's fees, resulting from, or in connection with Licensee's use of the Software. No action, regardless of form, arising out of the transactions under this Agreement may be brought by either party against the other more than two (2) years after the cause of action has accrued, except for actions related to unpaid fees.

9. UNITED STATES GOVERNMENT RESTRICTED RIGHTS. This section applies to all acquisitions of the Products by or for the United States government. By accepting delivery of the Products except as provided below, the government or the party procuring the Products under government funding, hereby agrees that the Products qualify as “commer-

cial" computer software as that term is used in the acquisition regulations applicable to this procurement and that the government's use and disclosure of the Products is controlled by the terms and conditions of this Agreement to the maximum extent possible. This Agreement supersedes any contrary terms or conditions in any statement of work, contract, or other document that are not required by statute or regulation. If any provision of this Agreement is unacceptable to the government, Vendor may be contacted at Altair Engineering, Inc., 1820 E. Big Beaver Road, Troy, MI 48083-2031; telephone (248) 614-2400. If any provision of this Agreement violates applicable federal law or does not meet the government's actual, minimum needs, the government agrees to return the Products for a full refund.

For procurements governed by DFARS Part 227.72 (OCT 1998), the Software, except as described below, is provided with only those rights specified in this Agreement in accordance with the Rights in Commercial Computer Software or Commercial Computer Software Documentation policy at DFARS 227.7202-3(a) (OCT 1998). For procurements other than for the Department of Defense, use, reproduction, or disclosure of the Software is subject to the restrictions set forth in this Agreement and in the Commercial Computer Software - Restricted Rights FAR clause 52.227-19 (June 1987) and any restrictions in successor regulations thereto.

Portions of Altair's PBS Professional Software and Documentation are provided with RESTRICTED RIGHTS. Use, duplication, or disclosure by the Government is subject to restrictions as set forth in subdivision(c)(1)(ii) of the rights in the Technical Data and Computer Software clause in DFARS 252.227-7013, or in subdivision (c)(1) and (2) of the Commercial Computer Software-Restricted Rights clause at 48 CFR52.227-19, as applicable.

10. CHOICE OF LAW AND VENUE. This Agreement shall be governed by and construed under the laws of the state of Michigan, without regard to that state's conflict of laws principles except if the state of Michigan adopts the Uniform Computer Information Transactions Act drafted by the National Conference of Commissioners of Uniform State Laws as revised or amended as of June 30, 2002 ("UCITA") which is specifically excluded. This Agreement shall not be governed by the United Nations Convention on Contracts for the International Sale of Goods, the application of which is expressly excluded. Each Party waives its right to a jury trial in the event of any dispute arising under or relating to this Agreement. Each party agrees that money damages may not be an adequate remedy for breach of the provisions of this Agreement, and in the event of such breach,

the aggrieved party shall be entitled to seek specific performance and/or injunctive relief (without posting a bond or other security) in order to enforce or prevent any violation of this Agreement.

11. [RESERVED]

12. Notice. All notices given by one party to the other under the Agreement or these Additional Terms shall be sent by certified mail, return receipt requested, or by overnight courier, to the respective addresses set forth in this Agreement or to such other address either party has specified in writing to the other. All notices shall be deemed given upon actual receipt.

Written notice shall be made to:

Altair: **Licensee Name & Address:**

Altair Engineering, Inc._____

1820 E. Big Beaver Rd_____

Troy, MI 48083_____

Attn: Tom M. PerringAttn: _____

13. TERM. For annual licenses, or Support provided for perpetual licenses, renewal shall be automatic for each successive year ("Renewal Term"), upon mutual written execution of a new Order Form. All charges and fees for each Renewal Term shall be set forth in the Order Form executed for each Renewal Term. All Software licenses procured by Licensee may be made coterminous at the written request of Licensee and the consent of Altair.

14. TERMINATION. Either party may terminate this Agreement upon thirty (30) days prior written notice upon the occurrence of a default or material breach by the other party of its obligations under this Agreement (except for a breach by Altair of the warranty set forth in Section 8 for which a remedy is provided under Section 10; or a breach by Licensee of Section 5 or Section 6 for which no cure period is provided and Altair may terminate this Agreement immediately) if such default or breach continues for more than thirty (30) days after receipt of such notice. Upon termination of this Agreement, Licensee must cease using the Software and, at Altair's option, return all copies to Altair, or certify it has destroyed all such copies of the Software and Documentation.

15. GENERAL PROVISIONS. Export Controls. Licensee acknowledges that the Products may be subject to the export control laws and regulations of the United States and other countries, and any amendments thereof. Licensee agrees that Licensee will not directly or indirectly export the Products into any country or use the Products in any manner except in compliance with all applicable U.S. and other countries export laws and regulations. **Notice**. All notices given by one party to the other under this Agreement shall be sent by certified mail, return receipt requested, or by overnight courier, to the respective addresses set forth in this Agreement or to such other address either party has specified in writing to the other. All notices shall be deemed given upon actual receipt. **Assignment**. Neither party shall assign this Agreement without the prior written consent of other party, which shall not be unreasonably withheld. All terms and conditions of this Agreement shall be binding upon and inure to the benefit of the parties hereto and their respective successors and permitted assigns. **Waiver**. The failure of a party to enforce at any time any of the provisions of this Agreement shall not be construed to be a waiver of the right of the party thereafter to enforce any such provisions. **Severability**. If any provision of this Agreement is found void and unenforceable, such provision shall be interpreted so as to best accomplish the intent of the parties within the limits of applicable law, and all remaining provisions shall continue to be valid and enforceable. **Headings**. The section headings contained in this Agreement are for convenience only and shall not be of any effect in constructing the meanings of the Sections. **Modification**. No change or modification of this Agreement will be valid unless it is in writing and is signed by a duly authorized representative of each party. **Conflict**. In the event of any conflict between the terms of this Agreement and any terms and conditions on a Licensee Purchase Order or comparable document, the terms of this Agreement shall prevail. Moreover, each party agrees any additional terms on any Purchase Order or comparable document other than the transaction items of (a) item(s) ordered; (b) pricing; (c) quantity; (d) delivery instructions and (e) invoicing directions, are not binding on the parties. In the event of a conflict between the terms of this Agreement, and the Additional Terms, the Agreement shall take precedence. **Entire Agreement**. This Agreement, the Additional Terms, and the Order Form(s) attached hereto constitute the entire understanding between the parties related to the subject matter hereto, and supersedes all proposals or prior agreements, whether written or oral, and all other communications between the parties with respect to such subject matter. This Agreement may be executed in

one or more counterparts, all of which together shall constitute one and the same instrument. **Execution**. Copies of this Agreement executed via original signatures, facsimile or email shall be deemed binding on the parties.

Index

A

Accounting
  job arrays 226
accounting 200
ACCT_TMPDIR 200
Administrator Guide 11
Advance reservation
  creation 183
advance reservation 181
AIX 236
  Large Page Mode 203
Altering
  job arrays 221
Ames Research Center ix
AOE 321
  using 323
API 4, 180
application licenses
  floating 41
  node-locked
    per-CPU 42
    per-host 42
    per-use 42
arrangement 51
Attribute
  account_string 90
  priority 84
  rerunnable 83
attributes
  modifying 115

B

Batch
  job 11
block 160
Boolean Resources 39

Built-in Resources 27

C

Changing
    order of jobs 125
Checking status
    of jobs 134
    of queues 138
    of server 137
Checkpointable 86
Checkpointing
    interval 86
    job arrays 226
    SGI MPI 272
checkpointing 120
chunk 38
CLI 11
Command line interface 11
Commands 3
commands
    and provisioning 327
comment 145
count_spec 185
credential 202
CSA 200
csh 15
Custom resources 36

D

DCE 201, 202
Dedicated Time 199
Default Resources 40
Deleting
    job array range 221
    job arrays 221
    subjob 221
Deleting Jobs 122
Deprecations 10
Destination
    specifying 77
devtype 238
directive 11, 21, 67, 68, 69, 70, 108, 178, 331, 332
Directives 31
directives 28
Display
    nodes assigned to job 144
    non-running jobs 143
    queue limits 145
    running jobs 142
    size in gigabytes 143
    size in megawords 144
    user-specific jobs 142
Distributed
    workload management 1

E

Email
    notification 81
euidevice 238
euilib 238
exclhost 51
exclusive 51
Executor 3
Exit Status
    job arrays 227

F

Fairshare
    job arrays 227
File
    output 165
    output and error 78
    rhosts 18
    specify name of 77
    staging 166
Files
    cshrc 15
    hosts.equiv 19
    login 15

pbs.conf 21, 111
profile 15
rhosts 19
xpbsrc 110
files
.login 15
.logout 15
floating licenses 41
free 51
freq_spec 185

G

Graphical user interface 11
group=resource 51
grouping 51
GUI 11

H

here document 35
hfile 238
Hitchhiker's Guide 242
Hold
or release job 119
Holding a Job Array 222
hostfile 238
HPC Basic Profile 273
HPC Basic Profile Job 274
HPC Basic Profile Server 273
HPCBP
Executable location 276
Job resources 280
Job submission requirements 276
Monitoring jobs 282
Password requirement 275
qsub command 278
qsub syntax 278
Submitting jobs 276
Unsupported commands 290
User account 275
HPCBP Job 274
HPCBP MOM 274
HPS
IP mode 236
US mode 236

I

IBM POE 236
identifier 33
Identifier Syntax 207
InfiniBand 265, 266
ensuring use 237
instance 181
instance of a standing reservation 181
instances
option 239
Intel 260
Intel MPI 260
examples 262
Interactive job submission
job arrays 208
Interactive-batch jobs 91
interval_spec 185
IP mode HPS 236

J

ja 200
Job
checkpointable 86
comment 145
dependencies 161
identifier 33
name 82
selecting using xpbs 155
sending messages to 123
sending signals to 124
submission options 74
tracking 156
Job Array

Attributes 209
identifier 206
range 206
States 210
Job Array Run Limits 225
Job Arrays 205
checkpointing 226
deleting 221
exit status 227
interactive submission 208
PBS commands 216
placement sets 228
prologues and epilogues 226
qalter 221
qdel 221
qhold 222
qmove 222
qorder 222
qrerun 223
qrls 222
qrun 223
qselect 224
run limits 225
starving 225
status 218
submitting 208
Job Arrays and xpbs 225
job container 200
Job Script 28
Job Submission Description Language 274
Job Submission Options 74
jobs
MPI 230
PVM 271
SMP 230
job-wide 39
JSDL 274

K

Kerberos 202
qsub -W cred=DCE 201
KRB5 202
krb5 202

L

Large Page Mode 203
Limits on Resource Usage 49
Linux job container 200
Listbox 95
LoadLeveler 236

M

man pages
SGI 16
MANPATH 16
Message Passing Interface 235
Modifying Job Attributes 115
MOM 3
Monitoring 2
Moving 222
jobs between queues 126
Moving a Job Array 222
MP_DEVTYPE 238
MP_EUIDEVICE 238
MP_EUILIB 238
MP_HOSTFILE 238
MP_INSTANCES 239
MP_PROCS 239
MPI 235
AIX and POE 236
HP-UX and Linux 245
Intel MPI 260
examples 262
MPICH_GM
rsh/ssh
examples 247
MPICH2 255, 266
examples 258, 267

MPICH-GM
MPD 250
examples 251
rsh/ssh 246
MPICH-MX
MPD 252
examples 254
rsh/ssh 248
examples 249
MVAPICH1 265
examples 266
SGI
Altix 272
MPI jobs 230
MPI, LAPI 236
MPICH 235
MPICH_GM
rsh/ssh
examples 247
MPICH2 255, 266
examples 258, 267
MPICH-GM
MPD 250
examples 251
rsh/ssh 246
MPICH-MX 248
MPD 252
examples 254
rsh/ssh 248
examples 249
MPI-OpenMP 232
mpirun 235
Intel MPI 260
MPICH2 255
MPICH-GM (MPD) 250
MPICH-GM (rsh/ssh) 246
MPICH-MX (MPD) 252
MPICH-MX (rsh/ssh) 248
MVAPICH1 265
MVAPICH2 266
mpirun.ch_gm 246
mpirun.ch_mx 248
mpirun.mpd 250, 252
MRJ Technology Solutions ix
MVAPICH1 265
examples 266

N

name 82
NASA
and PBS ix
Network Queueing System
nqs2pbs 331
Node Grouping
job arrays 228
Node Specification Conversion 65
Node specification format 65
nqs2pbs 12

O

OpenMP 232
Ordering job arrays 222
Ordering Job Arrays in the Queue 222
override 32

P

pack 51
Parallel
Virtual Machine (PVM) 271
password
single-signon 72
Windows 72
PBS commands
job arrays 216
PBS Environmental Variables 210
PBS_ARRAY_ID 210
PBS_ARRAY_INDEX 210
PBS_DEFAULT 21
PBS_DEFAULT_SERVER 110
PBS_DPREFIX 21

PBS_ENVIRONMENT 15, 21
pbs_hostn 12
PBS_JOBID 210
pbs_migrate_users 12
PBS_O_WORKDIR 21
pbs_password 12, 72
pbs_probe 12
pbs_rdel 12
pbs_rstat 12
pbs_rsub 12, 188
pbs_tclsh 12
pbsdsh 12, 180
pbsfs 12
pbsnodes 12
pbs-report 12
Peer Scheduling
  job arrays 227
per-CPU node-locked licenses 42
per-host node-locked licenses 42
per-use node-locked licenses 42
place statement 50
placement sets
  job arrays 228
POE 236
poe
  examples 243
Preemption
  job arrays 227
printjob 12
procs 239
PROFILE_PATH 18
Prologues and Epilogues
  job arrays 226
ProPack 200
provision 321
provisioned vnode 322
provisioning 323
  allowing time 328
  and commands 327
  AOE restrictions 325
  host restrictions 325
  requesting 326
  using AOE 323
  vnodes 322
PVM 271

Q

qalter 12, 105
  job array 221
qdel 12, 105
  job arrays 221
qdisable 12, 105
qenable 12, 105
qhold 12, 105, 119, 121
  job arrays 222
qmgr 12
qmove 12, 106, 126
  job array 222
qmsg 12, 106, 123, 224
qorder 12, 106, 125, 126
  job arrays 222
qrerun 12, 105
  job arrays 223
qrls 12, 105, 120, 121
  job arrays 222
qrun 12, 105
  job array 223
qselect 12, 112, 113, 154, 155
  job arrays 224
qsig 12, 105, 124
qstart 13, 105
qstat 13, 105, 118, 121, 125, 126, 133, 134, 135, 138, 139, 142, 143, 144, 145, 153, 154
qstop 13, 105
qsub 13, 14, 67, 70, 71, 74, 105, 160, 161, 202
  Kerberos 201
qsub options 74
qterm 13, 105
Queuing 1

Quick Start Guide xii

R

rcp 13
recurrence rule 184
Releasing a Job Array 222
report 200
requesting provisioning 326
Requeuing a Job Array 223
Reservation
    deleting 193
reservation
    advance 181, 183
    degraded 182
    instance 181
    Setting start time & duration 186
    soonest occurrence 182
    standing 181
        instance 181
        soonest occurrence 182
    standing reservation 184
    Submitting jobs 194
reservations
    time for provisioning 328
Resource Specification Conversion 67
Resource specification format 67
resource_list 75
resources 31
restrictions
    AOE 325
    provisioning hosts 325
resv_nodes 182
rhosts 18
run limits
    job arrays 225
Running a Job Array 223

S

scatter 51
Scheduler 4
Scheduling 2
    job Arrays 227
scp 13
Selection of Job Arrays 224
selection statement 38
Sequence number 206
Server 3
setting job attributes 32
SGI MPI 272
share 51
sharing 51
shell 28
SIGKILL 124
SIGNULL 124
SIGTERM 124
single-signon 72
Single-Signon Password Method 72
SMP jobs 230
soonest occurrence 182
spec 65
spec_list 65
stageout 76
staging
    Windows
        job arrays 215
Standing Reservation 181
standing reservation 181, 184
Starving
    job arrays 225
States
    job array 210
states 112, 155
Status
    job arrays 218
stepping factor 208
Subjob 205
Subjob index 206
submission options 74

Submitting a job array 208
Submitting a PBS Job 23
suffix 65
Suppressing job identifier 91
syntax
  identifier 207

T

Task Manager 180
TCL 93
TGT 202
time between reservations 198
TK 93
tm(3) 180
TMPDIR 22
tracejob 13
tracking 156

U

umask 160
Unset Resources 26
until_spec 185
US mode HPS 236
User Guide xii
user job accounting 200
username 18
  maximum 14

V

Viewing Job Information 139
Vnode Types 24
vnodes
  provisioning 322
vscatter 51

W

Wait for Job Completion 160
Widgets 95
Windows 16, 18
  job arrays
    staging 215
  password 72
  staging
    job arrays 215

X

xpbs 13, 106, 110, 112, 114
  buttons 105
  configuration 110
  job arrays 225
  usage 93, 124, 125, 153, 155, 164
xpbsmon 13
xpbsrc 110